

NAVAL POSTGRADUATE SCHOOL

Monterey, California



DISSERTATION

AD-A214 568

DYNAMIC FACTORIZATION IN LARGE-SCALE
OPTIMIZATION

by

Michael P. Olson

June 1989

Thesis Advisor: Gerald G. Brown

Approved for public release; distribution is unlimited

DTIC
ELECTRONIC
NOV 22 1989
S B D

89 11 20 108

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION			1b RESTRICTIVE MARKINGS		
2a SECURITY CLASSIFICATION AUTHORITY			3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public release; Distribution is unlimited		
2b DECLASSIFICATION/DOWNGRADING SCHEDULE					
4 PERFORMING ORGANIZATION REPORT NUMBER(S)			5 MONITORING ORGANIZATION REPORT NUMBER(S)		
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School		6b OFFICE SYMBOL (If applicable) 55	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School		
6c ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000			7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000		
8a NAME OF FUNDING/SPONSORING ORGANIZATION		8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
8c ADDRESS (City, State, and ZIP Code)			10 SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO	PROJECT NO	TASK NO
11 TITLE (Include Security Classification) DYNAMIC FACTORIZATION IN LARGE-SCALE OPTIMIZATION					
12 PERSONAL AUTHOR(S) OLSON, Michael Paul					
13a TYPE OF REPORT Ph.D. Dissertation		13b TIME COVERED FROM _____ TO _____		14 DATE OF REPORT (Year, Month, Day) 1989, June	
15 PAGE COUNT 185					
16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government					
17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Mathematical programming, linear programming, factorization, Dynamic Row Factorization		
FIELD	GROUP	SUB GROUP			
19 ABSTRACT (Continue on reverse if necessary and identify by block number) Factorization is an approach to linear programming (LP) in which the algebraic elements of the LP tableau are organized in such a way that a large portion of the tableau may be represented implicitly and generated from the remaining explicit part. In dynamic row factorization, the row structure of the LP model instance influences the algebraic structure of the tableau, and the dimension of the algebraic elements may change as the solution progresses. We present three algorithms motivated by this approach, each resulting from a different LP model row structure: generalized upper bound (GUB) rows, pure network rows and generalized network rows. We describe implementations of all three algorithms, specifying data structures for tableau and basis inverse representations and detailing procedures for manipulation and update of these representations.					
20 DISTRIBUTION STATEMENT OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED <input type="checkbox"/> SAME AS REPORT <input type="checkbox"/> OTHER			21 ABSTRACT SECURITY CLASSIFICATION Unclassified		
22a NAME OF RESPONSIBLE INDIVIDUAL Gerald G. Brown			22b TELEPHONE (include Area Code) 408-646-2140		22c MAILING ADDRESS 55Bw

Block 19. Abstract (cont)

Computational results are presented for a number of real-world models taken from a variety of applications and industries. From each model, one or more particular instances are solved by each of our three implementations and by a commercial-quality mathematical programming system. Previous research on related algorithms by others suggests that these algorithms are properly viewed as specialized approaches, useful only on narrow classes of problems. Our computational results strongly refute this view, and instead suggest that each algorithm is superior to the general simplex approach on a wide range of problem classes and structures.

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Approved for public release; distribution is unlimited.

Dynamic Factorization in
Large-Scale Optimization

by

Michael P. Olson
Commander, Supply Corps, United States Navy
B.S., United States Naval Academy, 1974
M.S., Naval Postgraduate School, 1988

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN OPERATIONS RESEARCH

from the

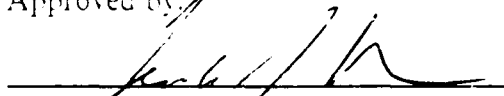
NAVAL POSTGRADUATE SCHOOL
June 1989

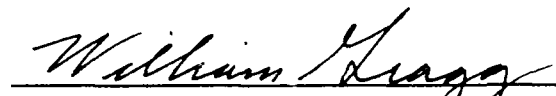
Author:

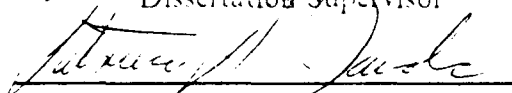


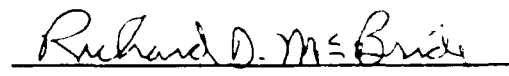
Michael P. Olson

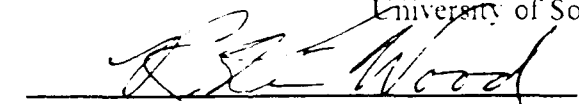
Approved by:


Gerald G. Brown
Professor of Operations Research
Dissertation Supervisor

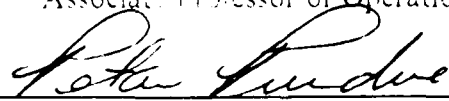

William Gragg
Professor of Mathematics


Patricia A. Jacobs
Professor of Operations Research

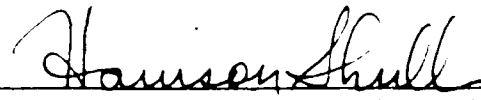

Richard D. McBride
Professor of Decision Systems
University of Southern California


R. Kevin Wood
Associate Professor of Operations Research

Approved by:


Peter Purdue, Chairman, Department of Operations Research

Approved by:


Harrison Shull, Academic Dean

ABSTRACT

Factorization is an approach to linear programming (LP) in which the algebraic elements of the LP tableau are organized in such a way that a large portion of the tableau may be represented implicitly and generated from the remaining explicit part. In dynamic row factorization, the row structure of the LP model instance influences the algebraic structure of the tableau, and the dimension of the algebraic elements may change as the solution progresses.

We present three algorithms motivated by this approach, each resulting from a different LP model row structure: generalized upper bound (GUB) rows, pure network rows and generalized network rows. We describe implementations of all three algorithms, specifying data structures for tableau and basis inverse representations and detailing procedures for manipulation and update of these representations.

Computational results are presented for a number of real-world models taken from a variety of applications and industries. From each model, one or more particular instances are solved by each of our three implementations and by a commercial-quality mathematical programming system. The characteristics of the four solvers are compared and contrasted. Previous research on related algorithms by others suggests that these algorithms are properly viewed as specialized approaches, useful only on narrow classes of problems. Our computational results strongly refute this view, and instead suggest that each algorithm is superior to the general simplex approach on a wide range of problem classes and structures.

TABLE OF CONTENTS

I.	INTRODUCTION	1
	A. INTRODUCTION	1
	B. SURVEY OF RELATED LITERATURE	3
II.	MUTUAL PRIMAL-DUAL METHOD	8
	A. INTRODUCTION	8
	B. PRIMAL PROBLEM STATEMENT (PLP)	8
	C. OBTAINING FEASIBILITY	10
	D. FROM FEASIBILITY TO OPTIMALITY	16
	E. BLOCKS	18
	F. THE PRIMAL ALGORITHM AND A SUPPORTING BASIC TAB- LEAU	19
	G. DUAL LINEAR PROBLEM (DLP)	24
	H. RESOLUTION OF BLOCKING	33
	I. RELATIONSHIP BETWEEN PRIMAL-DUAL ALGORITHM AND SIMPLEX METHOD	37
III.	IMPLEMENTATION DESIGN OVERVIEW	42
	A. INTRODUCTION	42
	B. DESIGN CONSIDERATIONS	42
	C. DESIGN TEMPLATE	43
	D. SEQUENCING COMPUTATIONS TO EXPLOIT COMMON SUB- EXPRESSIONS IN COLUMN GENERATION	47
	E. SEQUENCING COMPUTATIONS TO EXPLOIT COMMON SUB- EXPRESSIONS IN ROW GENERATION	49

IV.	FACTORIZATION	61
A.	INTRODUCTION	61
B.	THE FACTORED TABLEAU	62
C.	BENEFITS OF FACTORIZATION	67
V.	GENERAL IMPLEMENTATION TOOLS	70
A.	INTRODUCTION	70
B.	THE FACTORED TABLEAU	70
C.	ALGORITHM OVERVIEW	72
D.	IMPLEMENTATION CONVENTIONS	73
	1. The Tableau	73
	2. The Explicit Transformation Kernel	79
	3. The Factored Kernel	80
E.	COMPLETE ALGORITHM DESCRIPTION	81
VI.	FACTORIZATION OF GENERALIZED UPPER BOUND ROWS . . .	95
A.	INTRODUCTION	95
B.	THE FACTORED TABLEAU	96
C.	SEQUENCING COMPUTATIONS TO EXPLOIT COMMON SUB- EXPRESSIONS IN COLUMN GENERATION	98
D.	SEQUENCING COMPUTATIONS TO EXPLOIT COMMON SUB- EXPRESSIONS IN ROW GENERATION	102
E.	DATA STRUCTURES	105
F.	FACTORED KERNEL UPDATE ACTIONS	106
VII.	FACTORIZATION OF PURE NETWORK ROWS	109
A.	INTRODUCTION	109
B.	THE FACTORED TABLEAU	112

C.	SEQUENCING COMPUTATIONS TO EXPLOIT COMMON SUB- EXPRESSIONS IN COLUMN GENERATION	114
D.	SEQUENCING COMPUTATIONS TO EXPLOIT COMMON SUB- EXPRESSION IN ROW GENERATION	117
E.	DATA STRUCTURES	119
F.	SOLVING LINEAR SYSTEMS	121
G.	FACTORED KERNEL UPDATE	128
VIII.	FACTORIZATION OF GENERALIZED NETWORK ROWS	134
A.	INTRODUCTION	134
B.	THE FACTORED TABLEAU	137
C.	SEQUENCING COMPUTATIONS TO EXPLOIT COMMON SUB- EXPRESSIONS IN COLUMN GENERATION	138
D.	SEQUENCING COMPUTATIONS TO EXPLOIT COMMON SUB- EXPRESSIONS IN ROW GENERATION	138
E.	DATA STRUCTURES	138
F.	SOLVING LINEAR SYSTEMS	139
G.	FACTORED KERNEL UPDATE	141
IX.	COMPUTATIONAL RESULTS	145
A.	INTRODUCTION	145
B.	TEST PROBLEMS	145
C.	METHODOLOGY	149
D.	COMPUTATIONAL RESULTS	151
X.	CONCLUSIONS	163
	LIST OF REFERENCES	168
	INITIAL DISTRIBUTION LIST	174

LIST OF TABLES

5.1	Indices of Tableau Regions	75
5.2	Secondary and Tertiary Tableau Exchanges	94
9.1	Description of Test Suite Models	145
9.2	Summary of Problem Suite Dimensions	158
9.3	Solution Times in CPU Seconds	159
9.4	Number of Elements in Explicit Transformation Kernel Representa- tion of Optimality or at Failure	160
9.5	Summary of the Number of Binding Explicit Constraints at Optimality	161
9.6	$f(s, t)$ in Megabyte · seconds	162

LIST OF FIGURES

2.1	Primal Block in Column k_3	34
2.2	Subproblem (1)	34
2.3	Dual Block in Subproblem (1)	35
2.4	Subproblem (2)	35
2.5	Primal Block in Subproblem (2)	36
2.6	Subproblem 3	37
3.1	Bump Triangular Form of A_{11}	53
3.2	Spikes within a Bump	54
3.3	LU Decomposition of A_{11}	55
7.1	A Triangulated F_{11}	124
7.2	A Basis Graph $\mathcal{G}_{F_{11}}$	125
7.3	An Alternate Graph Paradigm $\hat{\mathcal{G}}_{F_{11}}$	126
7.4	The Implementation Paradigm for $\mathcal{G}_{F_{11}}$	128
7.5	Initialization of Node 9 Prior to Incorporation into $\mathcal{G}_{F_{11}}$	131
8.1	Near-Triangulated Simplex Basis Corresponding to (GNE)	136
8.2	A Sample Nearly-Triangulated (GNE) Simplex Basis Component	136
8.3	"One-tree" Subgraph	137

ACKNOWLEDGMENTS

I extend my sincere appreciation to those who have significantly influenced this research effort, namely:

Rick Rosenthal, who first sparked my interest in optimization,

Kevin Wood, whose remarkable insights helped light the path, and most especially to

Jerry Brown, a brilliant researcher, an inspiring advisor and one of the finest people I've had the pleasure to know.

I also wish to thank those whose support was vital in bringing this effort to fruition:

Jan Evans, my patient and enduring typist and friend.

Steve Pilnick, my kindred soul in this long and sometimes tortuous path.

The X-System, which was often cruel but always sorry.

Lastly, but most importantly, I thank my daughters, Elizabeth and Katie, for their patience and understanding during this effort, and my wonderful wife Susan, who provided far more support than she is willing to admit. Her love and sharing make life a far more glorious and fulfilling experience, than it otherwise would be.

I. INTRODUCTION

A. INTRODUCTION

A recurring theme in the development of simplex-based algorithms for linear programming has been the identification and exploitation of special problem structure. Ideas as apparently disparate as the simplex method for bounded variables, primal and dual decomposition methods, pure and generalized network primal simplex algorithms and primal partitioning schemes may be unified to a degree by interpreting their development in this context.

The factorization approach due to Graves and McBride [1976] provides a unifying framework from which we may reinterpret many existing algorithms and, through the application of the common principles embodied in the approach, develop new algorithms. This approach has as its central thesis the idea of recognizing, isolating and exploiting special structures which may occur in a certain type of linear programming tableau. An example of such special structure is generalized network rows. Generalized network rows are naturally specified as a row structure in which each column has at most two nonzero elements within the rows.

This paper adopts the design principles and algorithmic structure suggested by Graves and McBride [1976] to develop algorithms that, while general in nature in the sense that each may be used to solve any linear programming problem instance, are strongly tailored to exploit a particular row structure. We call this approach "dynamic row factorization"; "row factorization" because we exploit the structure in the basic tableau which is induced by the row structure of the LP model instance, and "dynamic" because the dimension of the structure may vary (or even fail to be

present) as the solution progresses. In our setting, we require the row structure of the model instance to be specified prior to solving, and to remain fixed throughout the solution process. An extension of this approach is to allow the row structure of the model instance to vary as the problem is solved. While we do not develop an implementation of the second approach, we show that it is a conceptually simple extension of our work.

Each algorithm is developed by factoring the constraints of the LP model into two classes: those that have special structure (factored) and those that do not (explicit). This factoring of constraints induces a factored structure in the LP tableaux which may be exploited computationally. We treat each of three structures of factored constraints in this work: generalized upper bounds (GUB), pure networks and generalized networks.

We implement each of the three algorithms by integrating it within the structure of a state-of-the-art mathematical programming system: the X-System of Brown and Graves [1975]. We do so both to demonstrate the feasibility of implementing such methods and to determine whether or not such methods have practical value for the practitioner interested in solving real-world problems. Commercial implementations of similar GUB algorithms have generally failed to inspire enthusiasm and are apparently falling into disuse [Kennington [1978]]. While the first commercial implementation of a related "pure network with side constraints" algorithm is (to our knowledge) just now becoming available, reports of research implementations have generally supported the view that such algorithms have limited application. The reports of implementations of similar "generalized network with side constraint" algorithms have offered less promise than their pure network counterparts.

After we review the related literature, we provide a detailed presentation of the underlying tableau-based algorithm which forms the foundation of all our subsequent

work. We do so because the algorithm is not widely known and may be unfamiliar to the reader. We then review the general factorization approach of Graves and McBride [1976]. We provide a design template for our developmental approach, and then present the dynamic row factorization algorithms for GUB, pure network and generalized network row structures. Computational results are presented, and we then summarize our conclusions and suggest avenues for further research.

B. SURVEY OF RELATED LITERATURE

While the terms “partitioning” and “factorization” are frequently used interchangeably in the literature, we observe a distinction between the two approaches. We consider partitioning methods to be those that are based on special structure in the original problem instance. This structure in the problem instance need not induce special structure into the LP tableau, and in fact the method need not be tableau-based. This is in contrast to our view of factorization, in which the algorithm is based on special structure which occurs in the simplex basis and thus in the basic tableau. We thus classify Dantzig-Wolfe [1960] and Benders [1962] Decompositions and Rosen’s [1964] primal partitioning method as examples of partitioning methods.

Perhaps the first example of what we consider factorization is the treatment of simple upper bounds by Dantzig [1954] and, independently, by Charnes and Lemke [1954]. They observe that it is more efficient to enforce the “structural” simple upper bound constraints with logical tests within the algorithm rather than treat them explicitly along with other constraints. While not originally presented in the context of a formal tableau factorization, the approach is easily viewed as such and is consistent with the general approach.

The mutual primal-dual method of Graves [1965] focuses attention on the special role of nonnegativity constraints in linear programming. A clear distinction is drawn between the computational convenience of treating nonnegativity constraints implicitly rather than explicitly and the unambiguous mathematical equivalence of all problem constraints, structural or nonnegativity. Emphasizing the special importance of inequality constraints, the approach yields an elegant theory (see Graves [1987]) and, as we will see, efficient implementations. We view this algorithm as the first formal example of factorization.

Dantzig and Van Slyke [1967] extend the factorization approach applied earlier to simple upper bounds in a more structured manner in their treatment of generalized upper bounds (GUB). In a problem with p GUB constraints and m structural constraints, their approach requires a working basis inverse of dimension $(m + 1)$, a considerable savings when p is large.

Hartman and Lasdon [1972] specialized this approach to the multicommodity capacitated transshipment problem. In this case, the structure of the basic pure network columns introduces additional structure into the working basis, allowing further simplifications in basis representation and update techniques. Helgason and Kennington [1977] develop techniques for representing the working basis inverse in product form and provide graphic interpretation of the graph updates. Kennington [1977] reports an implementation of the algorithm.

McBride [1972] and Graves and McBride [1976] formalize and generalize the factorization approach. They view it as a unifying framework for tableau-based simplex specializations and illustrate this by developing a variation of the GUB algorithm of Dantzig and Van Slyke [1967] and a GUB algorithm for doubly coupled linear programs of Hartman and Lasdon [1970]. They present a new algorithm for the set partitioning linear programming problem and an equality-constrained

form of the pure network with side constraints model. McBride [1972] reports an implementation of a GUB factorization.

Schrage [1975] extends the approach of simple and generalized upper bounds by considering variable upper bounds (VUB), which are constraints of the form $x_j \leq x_k$, where x_k is said to be the variable upper bound of x_j . His algorithm allows the implicit representation of the VUB constraints by expressing VUB variables in terms of other basic columns. This permits the basis representation to be treated in two parts, one part a large matrix which changes infrequently and thus needs to be updated only occasionally, and the second part a small working basis which requires regular attention. Thus, computation and storage savings may be realized. Schrage [1978] extends these ideas to what he calls generalized variable upper bounds (GVUB) constraints, which arise frequently in models involving fixed charges.

Klingman and Russell [1975] sketch a factorization method for solving transportation problems with side constraints. They suggest techniques for performing simplex iterations and updating the problem representation. Chen and Saigal [1977] present a similar approach for solving capacitated network flow problems with additional linear constraints. Both the above presentations consider a graph-theoretic view of the basis update mechanism and allow the basis representation to be treated in two parts, a part which corresponds to a rooted spanning tree defined on the underlying graph, and a general working basis inverse. Glover, Karney, Klingman and Russell [1978] report an implementation of the Klingman and Russell design, but one which (curiously) only accommodates a single side constraint. McBride [1989] reports an implementation which requires the pure network rows to be equalities and allows more than one side constraint.

The problem of generalized network problems with side constraints is addressed by Hultz and Klingman [1976]. They present details for the simplex price-out, column generation and basis update. Hultz and Klingman [1978] report an implementation that (curiously) solves the "singularly constrained" generalized network problem. McBride [1989] reports an implementation that is not restricted to a single side constraint.

The factorization approach has been extended by the consideration of embedded structures. Glover and Klingman [1981] consider a linear program which contains embedded pure network structure, i.e., the pure network structure appears in only a subset of the rows and columns of the technological coefficient matrix of the problem. Their approach yields an algorithm similar in spirit to the algorithms for the pure network with side constraint model, but the presence of the "side variables" significantly complicates the basis representation and update. They report an implementation of the algorithm but (curiously) restrict the problem suite to problems having no complicating variables.

McBride [1985] treats the problem of linear programs with embedded generalized network structure. He presents methods for pricing, column generation, basis representation update and data structures. A successful implementation is reported which is approximately five times faster than MINOS [1977] for the models tested.

Interest in developing algorithms to solve problems with special substructures has been accompanied by work to identify such substructures in problem instances. Greenberg and Rarick [1974] and Brown and Thomen [1980] develop algorithms to identify GUB sets. Brown and Wright [1984] develop algorithms for identifying pure network constraint substructures. Brown, McBride and Wood [1985] present a method for locating generalized network structures, both embedded and row-only structures.

Todd [1983] examines factorization from a geometric standpoint and constructs a geometric interpretation which is in large measure equivalent to the algebraic development of Graves and McBride [1976].

II. MUTUAL PRIMAL-DUAL METHOD

A. INTRODUCTION

This chapter presents the mutual primal-dual linear programming method introduced by Graves [1965] which provides the algorithmic framework and notational conventions for the research which follows.

We begin with a complete algebraic development of the primal algorithm followed by a less detailed symmetric discussion of the corresponding dual algorithm. We conclude with a unified treatment of the two which establishes the theoretical importance of the algorithm and justifies its use as the foundation of our specializations.

The following presentation scrupulously restates the Graves [1965] algorithm, incorporates later discussion by McBride [1972], and accommodates large-scale implementations by Brown and Graves [1975]. The view presented here is not available in standard reference texts, and is included in the interest of completeness.

B. PRIMAL PROBLEM STATEMENT (PLP)

The traditional statement of the linear programming (LP) problem is:

$$\begin{aligned} \text{(LP)} \quad \min_y : & \quad wy \\ \text{s.t.} \quad & \quad a_i y \leq r_i, \quad i = 1, \dots, m \\ & \quad e_j y \geq 0, \quad j = 1, \dots, n, \end{aligned}$$

where y is an n -vector of decision variables, w is an n -vector of cost coefficients, each a_i is an n -vector of technological transformation coefficients, each r_i is a scalar right-hand side coefficient, and e_j is the j^{th} unit vector. While this statement of

the problem is clear and unambiguous, there are reasons for preferring an alternative. The insistence upon drawing a formal distinction between the "structural" constraints $a_i y \leq r_i$ and the "nonnegativity" constraints $e_j y \geq 0$ obscures the mathematical structure of the problem by suggesting that the two types of constraints are inherently different. Certainly the exploitation of the special structure of the $e_j y \geq 0$ constraints leads to computational efficiencies in the implementation of the algorithm. However, in the theoretical development of the algorithm, we prefer to treat them simply as general inequality constraints.

In order to achieve a consistent form, we rewrite the nonnegativity constraints as $-e_j y \leq 0$ and group them with the structural constraints. The problem statement then becomes:

$$\begin{aligned} \text{(PLP)} \quad \min_y : \quad & w y \\ \text{s.t. :} \quad & a_i y \leq r_i, \quad i = 1, \dots, m+n, \end{aligned}$$

where $w y$ is called the extremal function. The constraints of (PLP) define a set of feasible solutions which we shall call the feasible set, F :

$$F \triangleq \{y \in R^n \mid a_i y \leq r_i, \quad i = 1, \dots, m+n\}.$$

Since F is the intersection of a finite number of closed half-spaces, F is itself a closed set. If F is nonempty and bounded, then an optimal solution to (PLP) will occur at an extreme point of F .

A point $y^0 \in F$ is said to be a *feasible point* or *feasible solution*. If, for constraint i , $a_i y^0 \leq r_i$, constraint i is *satisfied* at y^0 , and the quantity $r_i - a_i y^0$ is the *slack* in constraint i at y^0 . If, on the other hand, for constraint i , $a_i y^0 > r_i$, constraint i is *violated* at the point y^0 , the magnitude of the violation being $a_i y^0 - r_i$ (the negative of the slack).

A point $y^0 \in R^n$ is defined to be a *basic solution* of (PLP) if there exists an independent subset $\{a_{i_1}, a_{i_2}, \dots, a_{i_n}\}$ of $\{a_1, a_2, \dots, a_{m+n}\}$ such that $a_{i_j} y^0 = r_{i_j}$, for $j = 1, \dots, n$. Such an independent subset $\{a_{i_1}, a_{i_2}, \dots, a_{i_n}\}$ of $\{a_1, a_2, \dots, a_{m+n}\}$ is a *basis* for R^n , and y^0 is the (basic) solution to the system $a_{i_j} y^0 = r_{i_j}$, $j = 1, \dots, n$.

Each basic solution of (PLP) corresponds to an *extreme point* of F and for each extreme point of F there is at least one corresponding basic solution of (PLP). Since there are at most $\binom{m+n}{n}$ ways of choosing an independent subset of n vectors from $\{a_1, a_2, \dots, a_{m+n}\}$, the number of basic solutions of (PLP) and thus the number of extreme points of F is finite. Hence, (PLP) can be solved by searching among its basic solutions. The algorithm to be developed here will implicitly enumerate the set of basic solutions of (PLP) and terminate in one of three states:

1. $F = \emptyset$
(no feasible solution exists) ;
2. the extremum is unbounded
(for every real number α there is a point $y^0 \in F$ with $wy^0 < \alpha$); or
3. there exists at least one optimal solution
(a point $y^* \in F$ with $wy^* \leq wy \forall y \in F$).

We will first consider the problem of finding a basic *feasible* solution to (PLP). Having achieved this, we will then consider the task of finding a basic *feasible* solution which is also *optimal*.

C. OBTAINING FEASIBILITY

Since we have included the nonnegativity constraints $-e_j y \leq 0$, $j = 1, \dots, n$ in our structural constraints $a_i y \leq r_i$, $i = 1, \dots, m+n$, and since the origin is the unique solution to the independent system:

$$-e_j y = 0, \quad j = 1, \dots, n,$$

the origin is a basic solution that is always immediately available for (PLP).

Let y^0 be any basic solution to (PLP). By definition, there are at least n linearly independent constraints which are exactly binding at y^0 . Among the m remaining constraints, typically some will be satisfied at y^0 and others will be violated. Our strategy will be to focus on one violated constraint at a time, which we will call the *target* constraint. Moving from one basic solution to another, we will attempt to reduce the violation of the target constraint until it becomes satisfied. We will restrict our choices of basic solutions in such a way that all constraints that are already satisfied at y^0 remain satisfied at each subsequent basic solution. Once the target constraint becomes satisfied, we then select some other violated constraint as the new target constraint, and repeat the process. We proceed until either all constraints are satisfied and we have obtained a basic feasible solution, or we find a violated constraint which cannot be satisfied, in which case we conclude that no basic feasible solution exists.

To formalize these ideas, define $S(y^0)$ to be the set of indices of all constraints satisfied at a basic solution y^0 :

$$S(y^0) = \{1 \leq i \leq m + n \mid a_i y^0 \leq r_i\}.$$

Of course, $|S(y^0)| \geq n$.

Suppose constraint k is violated at the basic solution y^0 . Then $a_k y^0 > r_k$ and $k \notin S(y^0)$. A necessary condition for the existence of a basic feasible solution is that there exists a basic solution y with:

$$S(y) \supset S(y^0) \quad (2.1)$$

and

$$a_k y < a_k y^0. \quad (2.2)$$

If there exists no such basic solution y satisfying (Eq. 2.1) and (Eq. 2.2) we conclude that $F = \emptyset$. Thus, we may restrict our attention to basic solutions satisfying (2.1) and (2.2).

Let $\{a_{i_1}, a_{i_2}, \dots, a_{i_n}\}$ be a basis for R^n at y^0 . For notational convenience we will partition the constraints into two sets, those that are basic at y^0 and those that are nonbasic at y^0 :

$$B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} a_{i_1} \\ a_{i_2} \\ \vdots \\ a_{i_n} \end{bmatrix}; \quad f = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix} = \begin{bmatrix} r_{i_1} \\ r_{i_2} \\ \vdots \\ r_{i_n} \end{bmatrix} \quad (2.3)$$

$$D = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_m \end{bmatrix} = \begin{bmatrix} a_{i_{n+1}} \\ a_{i_{n+2}} \\ \vdots \\ a_{i_{n+m}} \end{bmatrix}; \quad g = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_m \end{bmatrix} = \begin{bmatrix} r_{i_{n+1}} \\ r_{i_{n+2}} \\ \vdots \\ r_{i_{n+m}} \end{bmatrix} \quad (2.4)$$

Define \mathcal{B} and \mathcal{D} to be the set of indices of the rows of B (basic constraints) and D (nonbasic constraints) respectively. Using this notation, the current basic solution y^0 may be expressed as $By^0 = f$, and since the rows of B are by definition linearly independent, B^{-1} exists and $y^0 = B^{-1}f$.

Let y be any other point of R^n . Then y can be expressed as:

$$y = y^0 + (y - y^0).$$

We have chosen this representation since at y^0 the basic constraints are satisfied exactly and thus it is the direction vector $(y - y^0)$ of any proposed move from y^0 to y that determines whether or not a given basic constraint will remain satisfied at y . (Of course, it is the magnitude of $(y - y^0)$ that is important in determining whether the nonbasic constraints remain satisfied as well.)

Now, for a given arbitrary basis $\{p^1, p^2, \dots, p^n\}$ of R^n and any point y , there must exist scalars $\lambda_1, \dots, \lambda_n$ such that:

$$y = y^0 + (y - y^0) = y^0 + \sum_{j=1}^n \lambda_j p^j = y^0 + P\lambda.$$

Since the choice of P is arbitrary, let us choose $P = -B^{-1}$, which we call the *conjugate row basis*. Then:

$$By = B(y^0 + P\lambda) = B(y^0 - B^{-1}\lambda) = By^0 - \lambda = f - \lambda,$$

and thus y satisfies the basic constraints $By \leq f$ if and only if $\lambda \geq 0$.

Thus, the set of points in R^n which satisfy the basic constraints $b_i y \leq f_i$ for $i \in B$ can be characterized as:

$$\left\{ y \in R^n \mid y = y^0 + \sum_{j \in B} \lambda_j p^j, \lambda_j \geq 0, j \in B \right\},$$

with $P = -B^{-1}$.

It then follows that every point y satisfying

$$S(y) \supseteq S(y^0), \tag{2.5}$$

can be expressed as

$$y = y^0 + P\lambda,$$

for some $\lambda \geq 0$ and $P = -B^{-1}$.

To address condition (2.2), assume that we have selected constraint k as our target constraint (and thus k is violated at y^0 , i.e., $d_k y^0 > g_k$). To reduce the violation of the target constraint, we seek a point y such that:

$$d_k y < d_k y^0, \quad (2.6)$$

holds. Since such a point must also satisfy (2.1), it must have a representation of the form $y = y^0 + P\lambda$ for some $\lambda \geq 0$. Replacing y by $y^0 + P\lambda$, we have:

$$d_k(y^0 + P\lambda) < d_k y^0,$$

which holds if and only if $d_k P\lambda < 0$. Since $\lambda \geq 0$, it follows that a necessary condition for the existence of a point y satisfying (2.5) and (2.6) is that at least one element of the vector $d_k P$ be negative, i.e., $d_k p^j < 0$ for some j , $1 \leq j \leq n$. If $d_k P \geq 0$, we conclude that $F = \emptyset$.

Suppose that $d_k p^l < 0$. Then (2.6) holds at all points of the form:

$$y = y^0 + \lambda_l p^l, \quad \lambda_l > 0.$$

The generic point $y^0 + \lambda_l p^l$ lies along an edge of the set of all points satisfying the constraints which are basic at y^0 . If there is more than one l for which $d_k p^l < 0$, any one can be chosen. For the purpose of exposition, we will designate the first such index encountered as l^* .

The violation in constraint k decreases linearly as λ_l increases. Constraint k becomes satisfied when:

$$d_k(y^0 + \lambda_l p^l) = g_k,$$

or when $\lambda_k = t$, with

$$t \triangleq \frac{(g_k - d_k y^0)}{d_k p^l}. \quad (2.7)$$

A geometric interpretation is that $y = y^0 + t p^l$ is the point at which the ray $\{y \in R^n \mid y = y^0 + \lambda_l p^l, \lambda_l \geq 0\}$ pierces the hyperplane $\{y \in R^n \mid d_k y = g_k\}$.

Define λ_l^* to be our ultimate choice for λ_l . Choosing t as the value for λ_l^* satisfies the target constraint k , but we are required by (2.5) to continue to satisfy all nonbasic constraints which are already satisfied at y^0 (if any). The choice of t may cause violations of such constraints. Thus, λ_l^* must also satisfy the condition:

$$d_j(y^0 + \lambda_l p^l) \leq g_j \quad \forall \quad j \in \mathcal{D} \cap S(y^0).$$

Writing this as

$$\lambda_l d_j p^l \leq g_j - d_j y^0,$$

we find that we must choose $\lambda_l \leq s$, where

$$s \triangleq \min_{j \in \mathcal{D} \cap S(y^0)} \left\{ \frac{(g_j - d_j y^0)}{d_j p^l} \mid d_j p^l > 0 \right\}. \quad (2.8)$$

If this set is empty, define $s = \infty$. Of course, it is possible for s to be equal to zero.

Thus, by choosing

$$\lambda_i^* \triangleq \min \{t, s\}, \quad (2.9)$$

where t and s are as defined in (2.7) and (2.8) respectively, we obtain the largest reduction in the violation of the target constraint consistent with the feasibility restriction (2.5) with respect to the chosen direction p^l .

The selection λ_i^* according to (2.9) leads to a new basic solution $y^1 = y^0 + \lambda_i^* p^l$. If $\lambda_i^* > 0$ at each step, the transitivity of (2.5) and (2.6) guarantees that no basic solution will be repeated before any given target constraint is satisfied, or until the conclusion is reached that $F = \emptyset$. At any given step, either the feasible set is shown to be empty, a basic solution is found which satisfies the target constraint or a positive reduction in the violation of the target constraint is achieved as the result of moving to another basic solution. Since the set of basic solutions is finite, the third alternative may be repeated a finite number of times before one of the first two alternatives occurs. If this constraint becomes satisfied, (2.5) guarantees that every subsequent solution will also satisfy the constraint. We may then select a new violated nonbasic constraint as the target constraint. Since there are a finite number of constraints, we will either discover a basic feasible solution or determine in a finite number of steps that none exists.

If $\lambda_i^* = 0$ at any step of the algorithm, we say that a block has occurred. Blocks will be discussed later in detail.

D. FROM FEASIBILITY TO OPTIMALITY

Once feasibility is achieved, say at y^0 , the process of proceeding to optimality can be thought of as minimizing over F the violation of the constraint $uy \leq uy^0 - \alpha$, where α is a sufficiently large positive constant.

Let y^0 be a basic feasible solution with $By^0 = f$ and $P = -B^{-1}$. Since every point $y \in R^n$ which satisfies $By \leq f$ may be written as $y = y^0 + P\lambda$ for some $\lambda \geq 0$, the value of the extremal function wy at such a point y is:

$$wy = w(y^0 + P\lambda) = wy^0 + wP\lambda = wy^0 + \sum_{j=1}^n \lambda_j wp^j.$$

Thus, a necessary condition for the existence of a point $y \in F$ such that $wy \leq wy^0$ is:

$$wp^j < 0 \text{ for some } j, \quad 1 \leq j \leq n.$$

If $wp^j \geq 0$, then y^0 must be a feasible, optimal solution to (PLP).

Suppose $wp^l < 0$. Since all constraints are satisfied at y^0 , the greatest reduction in the value of the extremal function in the direction p^l is achieved when λ_l^* is chosen to be s , where s is as defined in (2.8). If $s = \infty$, (PLP) has an unbounded extremum. If $0 < \lambda_l^* < \infty$, a positive reduction in the value of the extremal function can be achieved by moving from the basic feasible solution y^0 to the basic feasible solution $y = y^0 + \lambda_l^* p^l$. If $\lambda_l^* = 0$, a block is encountered.

Once a feasible solution has been obtained, (2.5) and (2.6) become equivalent to:

$$y \in F \quad . \quad (2.10)$$

$$wy < wy^0 \quad . \quad (2.11)$$

The transitivity of (2.10) and (2.11) ensures that no basic feasible solution will be repeated as long as $\lambda_l^* > 0$ at each step. At each iteration, either a basic feasible

solution is found to be optimal, the solution is found to be unbounded or a positive reduction in the value of the extremal function is achieved by moving to another basic feasible solution. Since the set of basic feasible solutions is finite and no basic feasible solution is repeated, the third alternative can only occur a finite number of times. Thus, if $\lambda_i^* > 0$ at each step and if a finite optimal solution exists, it will be found in a finite number of steps.

E. BLOCKS

Suppose that y^0 is a basic solution satisfying

$$By^0 = f.$$

where, as before, B is of dimension n by n and nonsingular and p^j is the j^{th} column of $P = -B^{-1}$. Then, y^0 is the unique point in R^n lying in the intersection of the n hyperplanes:

$$\{y \in R^n \mid b_i y^0 = f_i, \quad i = 1, \dots, n\}.$$

Suppose that p^i is a direction that either leads to a reduction in the violation of some target constraint or, if y^0 is feasible, a reduction in the value of the extremal function. Since the nonbasic constraints are of the form:

$$d_i y^0 \leq g_i, \quad i \in \mathcal{D},$$

a block occurs when, for at least one nonbasic constraint k ,

$$d_k y^0 = g_k.$$

and

$$d_k p^l > 0. \quad (2.12)$$

Thus, any movement away from y^0 in the direction p^l will result in violation of the nonbasic constraint $d_k y^0 \leq g_k$.

If $d_k y^0 = g_k$, then y^0 is in a sense "over-determined", and y^0 is said to be a "degenerate" solution. Geometrically, y^0 lies in the intersection of at least $n + 1$ hyperplanes. Algebraically, there is more than one basis that can be formed from the row vectors $\{a_1, a_2, \dots, a_{m+n}\}$ for which:

$$a_i, y^0 = r_i, \quad , \quad j = 1, \dots, n,$$

holds, and there is more than one basic solution which corresponds to the extreme point y^0 . A block is therefore encountered when an over-determined solution satisfies (2.12) in an "improving" direction (e.g., one that leads either to a reduction in infeasibility or, if feasible, to a reduction in the value of the extremal function). We will develop a method for dealing with blocks later.

F. THE PRIMAL ALGORITHM AND A SUPPORTING BASIC TABLEAU

The primal algorithm proceeds as follows:

1. Identify an initial basic solution. Notice that the origin satisfies

$$-I \cdot y^0 = 0.$$

and thus may always serve as the initial solution.

2. If the current solution is infeasible, select a violated primal constraint index k (i.e., an index k for which $g_k - d_k y^0 < 0$). This requires the quantity:

$$g - Dy^0 .$$

3. Within the target row, select an element of the proper sign (negative, according to our convention) whose index, l , specifies a "transformation column". If the current solution is infeasible, this requires the quantity:

$$d_k P ,$$

with the transformation column satisfying $d_k p^l < 0$. If the current solution is feasible, this requires the quantity:

$$wP$$

with the transformation column satisfying $w p^l < 0$. If no such element exists, then the problem is infeasible (if the current solution is infeasible and $d_k P \geq 0$) or optimal (if the current solution is feasible and $wP \geq 0$). The task of selecting such an index l is commonly referred to as "pricing" or a "pricing strategy".

4. Compute t as in (Equation 2.7). If the current solution is feasible, assign $t = \infty$.
5. Compute s as in (Equation 2.8). This computation is commonly called a "ratio test", or a "minimum ratio test". This computation requires the quantities:

$$g - Dy^0 \text{ and } Dp^l.$$

6. Compute λ_i^* as in (Equation 2.9). If $\lambda_i^* = \infty$, the problem is unbounded.

7. Update the current solution according to the computation:

$$y^1 = y^0 + \lambda_l^* p^l.$$

8. Update the assignments of constraint indices to \mathcal{B} and \mathcal{D} , and update P .

9. Go to Step 2.

The only quantities needed at each step of the algorithm are the matrix DP , the column vector $g - Dy^0$ and the row vector wP . For notational convenience, we can define \hat{D} to be an $(m+1)$ by n matrix whose bottom row \hat{d}_{m+1} is w , and \hat{g} to be a $(m+1)$ dimension column vector whose bottom entry, \hat{g}_{m+1} , is zero. Then

$$\hat{g}_{m+1} - \hat{d}_{m+1}y^0 = 0 - wy^0 = -wy^0,$$

which is the negative of the extremal function value at the point y^0 . We can now conveniently display all the relevant information by forming the $(m+1)$ by $(m+1)$ matrix:

$$[\hat{D}P \mid \hat{g} - \hat{D}y]. \quad (2.13)$$

which we will call the *basic tableau*.

By computing the basic tableau in partitioned matrix form, we may isolate the important algebraic components required by this method. To do so, let us assume that at the current basic solution the basis consists of h structural constraints and $(n-h)$ nonnegativity constraints. Then (2.3) can be written as:

$$B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_h \\ b_{h+1} \\ b_{h+2} \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} a_{i_1} \\ a_{i_2} \\ \vdots \\ a_{i_h} \\ -e_{j_{h+1}} \\ -e_{j_{h+2}} \\ \vdots \\ -e_{j_n} \end{bmatrix} \quad \text{and} \quad f = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_h \\ f_{h+1} \\ f_{h+2} \\ \vdots \\ f_n \end{bmatrix} = \begin{bmatrix} r_{i_1} \\ r_{i_2} \\ \vdots \\ r_{i_h} \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

In partitioned matrix form,

$$B = \begin{pmatrix} \overbrace{\quad}^h & \overbrace{\quad}^{n-h} \\ A_{11} & A_{12} \\ 0 & -I \end{pmatrix} \begin{matrix} \} h \\ \\ \} n-h \end{matrix}, \quad (2.14)$$

and thus

$$P = -B^{-1} = \begin{pmatrix} \overbrace{\quad}^h & \overbrace{\quad}^{n-h} \\ -A_{11}^{-1} & -A_{11}^{-1}A_{12} \\ 0 & I \end{pmatrix} \begin{matrix} \} h \\ \\ \} n-h \end{matrix}$$

Similarly, (2.4) can be written as:

$$D = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_h \\ d_{h+1} \\ d_{h+2} \\ \vdots \\ d_m \end{bmatrix} = \begin{bmatrix} -e_{j_1} \\ -e_{j_2} \\ \vdots \\ -e_{j_h} \\ a_{i_{h+1}} \\ a_{i_{h+2}} \\ \vdots \\ a_{i_{n+m}} \end{bmatrix} \quad \text{and} \quad g = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_h \\ g_{h+1} \\ g_{h+2} \\ \vdots \\ g_m \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ r_{i_{h+1}} \\ r_{i_{h+2}} \\ \vdots \\ r_{i_{n+m}} \end{bmatrix}.$$

and in partitioned matrix form:

$$D = \left(\begin{array}{cc} \overbrace{\quad}^h & \overbrace{\quad}^{n-h} \\ -I & 0 \\ A_{21} & A_{22} \end{array} \right) \begin{array}{l} \} h \\ \\ \} m-h \end{array} . \quad (2.15)$$

Then

$$DP = -DB^{-1} = \left(\begin{array}{cc} \overbrace{\quad}^h & \overbrace{\quad}^{n-h} \\ A_{11}^{-1} & A_{11}^{-1}A_{12} \\ -A_{21}A_{11}^{-1} & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{array} \right) \begin{array}{l} \} h \\ \\ \} m-h \end{array} ,$$

and we shall call DP the *principal part* of the tableau. By partitioning $w = (w_1, w_2)$, $g^T = (g_1, g_2)^T$ and $y^0 = (y_1^0, y_2^0)$, the basic tableau (2.13) may be written in partitioned matrix form as:

$$\left(\begin{array}{ccc} \overbrace{\quad}^h & \overbrace{\quad}^{n-h} & \overbrace{\quad}^1 \\ A_{11}^{-1} & A_{11}^{-1}A_{12} & y_1^0 \\ -A_{21}A_{11}^{-1} & A_{22} - A_{21}A_{11}^{-1}A_{12} & g_2 - A_{21}y_1^0 - A_{22}y_2^0 \\ -w_1A_{11}^{-1} & w_2 - w_1A_{11}^{-1}A_{12} & -wy^0 \end{array} \right) \begin{array}{l} \} h \\ \\ \} m-h \\ \} l \end{array} . \quad (2.16)$$

Note that y_1^0 is displayed explicitly in the tableau (2.16). Also, $y_2^0 = 0$ since the corresponding nonnegativity constraints are basic and thus binding.

As the algorithm proceeds from one basic solution to another, (2.16) can be updated using a slight variant of the Gauss-Jordan transformation (which we will call a *pivot*). The transformation is as follows:

Let $[\hat{D}P|\hat{g} - \hat{D}y^0] = [v_{ij}]$ be the basic tableau at any basic solution y^0 . If $v_{s,l} = d_s p^l \neq 0$ then the basic tableau at the basic solution:

$$y^1 = y^0 + \left(\frac{v_{s,n+1}}{v_{s,l}} \right) p^l = y^0 + \left(\frac{(q_s - d_s y^0)}{d_s p^l} \right) p^l.$$

written as

$$[\hat{D}'P' \mid \hat{g}' - \hat{D}'y'] = [v'_{ij}],$$

is calculated as:

general elements $(i \neq s, j \neq l)$:

$$v'_{ij} = v_{ij} - \frac{v_{s,j}v_{i,l}}{v_{s,l}},$$

pivotal column $(i \neq s, j = l)$:

$$v'_{il} = \frac{-v_{il}}{v_{s,l}},$$

pivotal row $(i = s, j \neq l)$:

$$v'_{s,j} = \frac{v_{s,j}}{v_{s,l}},$$

pivotal element $i = s, j = l$

$$v'_{s,l} = \frac{1}{v_{s,l}}.$$

After applying the transformation to (2.16), row and column interchanges may be necessary to restore the explicit structure of B in (2.14) and D in (2.15). A proof of this may be found in Graves [1987], who also extends the basic algorithm to include free variables, equality constraints and bounded variables.

G. DUAL LINEAR PROGRAM (DLP)

Corresponding to (PLP) is the following linear program called its dual:

$$\begin{aligned} \text{(DLP)} \quad \max_x : \quad & xr \\ \text{s.t.} : \quad & xA \leq w \\ & xI \leq 0. \end{aligned}$$

The relationship between (DLP) and (PLP) is as follows:

1. if (PLP) is infeasible, then (DLP) is either infeasible or unbounded.
2. if (PLP) is unbounded, then (DLP) is infeasible.
3. if (PLP) possesses a finite optimal solution y^* , then (DLP) also possesses a finite optimal solution x^* and $wy^* = x^*r$.

See Graves [1987] for a proof of this.

Solving (DLP) directly has important advantages in certain problems, and we will show that its solution plays an important role in dealing with blocks in (PLP). We will thus develop a dual algorithm in a manner similar to that for (PLP), sparing some of the symmetrical detail where possible.

Let x_o be a basic solution for (DLP). Then there exists an independent subset $\{a^{j_1}, a^{j_2}, \dots, a^{j_m}\}$ of $\{a^1, a^2, \dots, a^{m+n}\}$ such that $x_o a^{j_i} = w^{j_i}$, $i = 1, \dots, m$, and this linearly independent subset will be called the basis for R^m at x_o .

Partitioning the dual constraints into those which are basic at x_o and those which are nonbasic at x_o yields:

$$\begin{aligned} T &= (t^1, t^2, \dots, t^m) = (a^{j_1}, a^{j_2}, \dots, a^{j_m}) \\ u &= (u^1, u^2, \dots, u^m) = (u^{j_1}, u^{j_2}, \dots, u^{j_m}), \end{aligned} \quad (2.17)$$

and

$$\begin{aligned} K &= (k^1, k^2, \dots, k^n) = (a^{j_{m+1}}, a^{j_{m+2}}, \dots, a^{j_{m+n}}) \\ v &= (v^1, v^2, \dots, v^n) = (u^{j_{m+1}}, u^{j_{m+2}}, \dots, u^{j_{m+n}}). \end{aligned} \quad (2.18)$$

Define \mathcal{T} and \mathcal{K} to be the set of indices of the columns of T (basic dual constraints) and K (nonbasic dual constraints), respectively. The current basic solution x_o may then be expressed as

$$x_o = uT^{-1},$$

and any other point x in R^m may be expressed as

$$x = x_o + (x - x_o).$$

For a given arbitrary basis $\{q_1, q_2, \dots, q_m\}$ of R^m , any such point x may be represented as:

$$x = x_o + \sum_{i=1}^m q_i \psi^i = x_o + \psi Q.$$

The current basic constraints are $x_o a^{j_i} = u^{j_i}, i = 1, \dots, m$ and for these constraints to remain satisfied at the generic point $x = x_o + \psi Q$, we must have $[x a^{j_i} \leq u^{j_i}, i = 1, \dots, m]$. Choosing $Q = T^{-1}$ as a convenient basis for R^m at x_o , we then have:

$$xT = (x_o + \psi Q)T = x_o T + \psi QT = u + \psi,$$

and thus we must have $\psi \leq 0$.

Now consider a violated dual nonbasic constraint t , where a violation means

$$x_o k^t > v^t.$$

To reduce the violation, we seek a point x at which

$$x k^t < x_o k^t.$$

holds. Such a point must also satisfy the basic constraints, and thus the condition for a reduction in the violation of the dual target constraint is:

$$xk^t = (x_o + \psi Q)k^t = x_o k^t + \psi Qk^t < x_o k^t,$$

which implies that

$$\psi Qk^t < 0.$$

Since $\psi \leq 0$, a necessary condition for constraint t to be satisfied is that there exists an i with $q_i k^t > 0$. If none of the elements of $q_i k^t$ are positive, we conclude that the dual constraints are inconsistent.

The dual algorithm proceeds along the edges

$$x = x_o + \nu^j q_i,$$

from basic solution to basic solution. We define

$$S(x_o) = \{1 \leq j \leq m+n \mid x_o a^j \leq u^j\}.$$

and insist that

$$S(x) \supset S(x_o).$$

To satisfy this condition, we consider the effect of moving along the edge of a general nonbasic dual constraint k^t which is currently satisfied at x_o . We must have

$$xk^j = (x_o + \psi^i q_i)k^j = x_o k^j + \psi^i q_i k^j \leq v^j.$$

Since the constraint k^j is satisfied at x_o , we have

$$x_o k^j \leq v^j \Rightarrow v^j - x_o k^j \geq 0,$$

and since $\psi^i \leq 0$, if $q_i k^j < 0$ then as ψ^i decreases in value we approach the boundary of the constraint. If $v^j - x_o k^j = 0$ then $q_i k^j < 0$ causes an immediate violation if $\psi^i < 0$. Thus, a block has occurred.

We have shown that our choice for ψ^i , denoted ψ_\star^i , should be

$$\psi_\star^i = \max \{b, c\}, \quad (2.19)$$

where

$$b \triangleq (v^i - x_o k^i)/q_i k^i. \quad (2.20)$$

and

$$c \triangleq \max_{j \in K} \left\{ (v^j - x_o k^j)/q_i k^j \mid v^j - x_o k^j \geq 0, \quad q_i k^j < 0 \right\}. \quad (2.21)$$

Once dual feasibility has been achieved, we wish to maximize:

$$xr = x_o r + \psi(Qr).$$

Since $\psi \leq 0$, a necessary condition to achieve an increase in the value of the extremal function is that there exists an i such that $q_i r < 0$. Hence, when $q_i r \geq 0$, we conclude that x_o is dual optimal.

The dual algorithm then proceeds as follows:

1. Identify an initial basic solution. Notice that the origin satisfies

$$x^0 \cdot I = 0,$$

and thus may always serve as the initial solution.

2. If the current solution is infeasible, select a violated dual constraint index t (i.e., an index t for which $v^t - x_0 k^t < 0$). This requires the quantity:

$$v - x_0 K.$$

Column t is then referred to as the "target column". If the current solution is feasible, the right-hand side column is designated the "target column".

3. Within the target column, select an element of the proper sign (positive, according to our convention) whose index, i , specifies a "transformation row". If the current solution is infeasible, this requires the quantity:

$$Q k^t,$$

with the transformation row satisfying $q_i k^t > 0$. If the current solution is feasible, this requires the quantity:

$$Q r.$$

with the transformation row satisfying $q_i r > 0$. If no such element exists, then the problem is infeasible (if the current solution is infeasible and $q_i k^t \leq 0$) or optimal (if the current solution is feasible and $q_i r \leq 0$). The task of selecting such an index i is commonly referred to as "pricing", or a "pricing strategy".

4. Compute k as in (Equation 2.20). If the current solution is feasible, assign $b = -\infty$.

5. Compute c as in (Equation 2.21). This computation is commonly called a "ratio test". This computation requires the quantities:

$$v - x_0 K \text{ and } Qk^i.$$

6. Compute ψ_\star^i as in (Equation 2.19). If $\psi_\star^i = \infty$, the problem is unbounded.
7. Update the current solution according to the computation:

$$x_1 = x_0 + \psi_\star^i q_i.$$

8. Update the assignments of constraint indices to \mathcal{T} and \mathcal{K} , and update Q .
9. Go to Step 2.

This development shows that at each step, the quantities QK , $(v - x_0 K)$ and Qr are needed to proceed with the dual algorithm.

To develop a matrix partitioned form of the dual tableau, we proceed as before. Assuming the dual basis consists of h structural constraints and $m - h$ nonnegativity constraints, we have:

$$\begin{aligned} T &= (t^1, t^2, \dots, t^n) = (a^{j_1}, a^{j_2}, \dots, a^{j_h}, e^{i_{h+1}}, e^{i_{h+2}}, \dots, e^{i_m}) \\ u &= (u^1, u^2, \dots, u^m) = (u^{j_1}, u^{j_2}, \dots, u^{j_h}, 0, 0, \dots, 0), \end{aligned}$$

so

$$u = (u^1, u^2) = (u^1, 0).$$

The nonbasic constraints are then:

$$\begin{aligned} K &= (k^1, k^2, \dots, k^n) = (e^{i_1}, e^{i_2}, \dots, e^{i_h}, a^{j_{h+1}}, a^{j_{h+2}}, \dots, a^{j_n}) \\ v &= (v^1, v^2, \dots, v^n) = (0, 0, \dots, 0, u^{j_{h+1}}, u^{j_{h+2}}, \dots, u^{j_n}). \end{aligned}$$

so $v = (v^1, v^2) = (0, w^2)$.

The matrix partitioned form of the basis is then

$$T = \left(\begin{array}{cc} \overbrace{\quad}^h & \overbrace{\quad}^{m-h} \\ A_{11} & 0 \\ A_{21} & I \end{array} \right) \begin{array}{l} \} h \text{ ,} \\ \} m - h \text{ ,} \end{array}$$

and with the choice of $Q = T^{-1}$

$$Q = \left(\begin{array}{cc} \overbrace{\quad}^h & \overbrace{\quad}^{m-h} \\ A_{11}^{-1} & 0 \\ -A_{21}A_{11}^{-1} & I \end{array} \right) \begin{array}{l} \} h \text{ ,} \\ \} m - h \text{ ,} \end{array}$$

with the remaining constraints forming

$$K = \begin{bmatrix} I & A_{12} \\ 0 & A_{22} \end{bmatrix} .$$

The principal part of the dual tableau is:

$$\begin{aligned} QK &= \begin{bmatrix} A_{11}^{-1} & 0 \\ -A_{21}A_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} I & A_{12} \\ 0 & A_{22} \end{bmatrix} \\ &= \begin{bmatrix} A_{11}^{-1} & A_{11}^{-1}A_{12} \\ -A_{21}A_{11}^{-1} & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{bmatrix} , \end{aligned}$$

which we find to be exactly the principal part of the primal tableau, so

$$DP = QK .$$

The quantities $(v - x_i K)$ are

$$v - x_i K = v - (uQ)K$$

$$\begin{aligned}
&= v - uDP \\
&= (0, w^2) \begin{bmatrix} -A_{11}^{-1} & -A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix} - uDP \\
&= vP - uDP \\
&= (v - uD)P \\
&= \left\{ (0, w^2) - (w^1, 0) \begin{bmatrix} -I & 0 \\ A_{21} & A_{22} \end{bmatrix} \right\} P \\
&= \left\{ (0, w^2) - (-w^1, 0) \right\} P \\
&= wP.
\end{aligned}$$

The quantity Qr is:

$$\begin{aligned}
Qr &= Q \begin{pmatrix} r_1 \\ r_2 \end{pmatrix} \\
&= Q \left\{ \begin{pmatrix} r_1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ r_2 \end{pmatrix} \right\} \\
&= Q \left\{ g + \begin{bmatrix} I & A_{12} \\ 0 & A_{22} \end{bmatrix} \begin{pmatrix} 0 \\ r_2 \end{pmatrix} \right\} \\
&= Q \{g + Kf\} \\
&= Qg + QKf \\
&= \begin{bmatrix} A_{11}^{-1} & 0 \\ -A_{21}A_{11}^{-1} & I \end{bmatrix} \begin{pmatrix} 0 \\ r_2 \end{pmatrix} + DPf \\
&= \begin{pmatrix} 0 \\ r_2 \end{pmatrix} + DPf \\
&= g + D(Pf) \\
&= g - Dy^0.
\end{aligned}$$

We thus find that the quantities required for the dual tableau ($QK, v - x_0K$ and Qr) are exactly the quantities required for the primal tableau (DP, wP and $g - Dy^0$). Therefore, the primal and dual algorithms use the same tableau. Reviewing the summaries of the primal and dual algorithms, we see the strong symmetry between

the two. Operating on a single tableau, the primal algorithm identifies a target row, selects a transformation column, performs a ratio test, and performs a pivot update, while the dual algorithm identifies a target column, selects a transformation row, performs a ratio test and performs a pivot update. This remarkable symmetry allows us not only to perform either algorithm on a single tableau, but to switch between one algorithm and the other as often as we choose. This fact has important implications for our implementation and it enables us to deal with blocking in a systematic and consistent way.

H. RESOLUTION OF BLOCKING

The resolution of blocking in either the primal or dual algorithm will be accomplished by shifting to the alternate algorithm when blocking occurs. The alternate algorithm is applied to a subproblem of the original and at worst we are lead to a contracting sequence of problems to which we alternately apply the primal and dual algorithm. A strict contraction is assured, and thus in at most a finite number of steps resolution is achieved.

We will demonstrate this procedure by assuming that we have started with the primal algorithm. When a block occurs, assume we have rearranged the rows so that the zeros in the right-hand side column occur contiguously.

Let $k_1 = n + 1$ and $k_2 = t$ where t is the index of the target row (if the current solution is primal feasible, $k_2 = m + 1$, the extremal function row). Let k_3 be the pivot column which caused the block. The situation is shown in Figure 2.1.

Define Subproblem (1) as consisting of all the columns of the original problem, but only those rows with zeros in the right-hand side column, and as shown in Figure 2.2, "extremal function" row k_2 , the target row of the original primal problem. Row k_4 is the row containing the blocking element. Now apply the dual algorithm to

	k_3		k_1	
			+	
			+	
			+	
	+		0	
			0	
			0	
			+	
	-		-	k_2

Figure 2.1: Primal Block in Column k_3

	k_3		k_1	
	+		0	k_4
			0	
			0	
	-		-	k_2

Figure 2.2: Subproblem (1)

Subproblem (1). Because the right-hand side column of any potential pivot row is zero, the right-hand side column of the original primal problem is invariant to any pivot in Subproblem (1). Let v_{ij} denote the element in row i and column j of the tableau. We proceed with the dual algorithm on Subproblem (1) until one of the following situations occur (which may be after zero or more dual pivots):

1. $v_{k_2, k_3} \geq 0$. A pivot in column k_3 no longer reduces the violation of primal constraint k_2 , and thus the primal block has been eliminated. We thus return to the original primal problem and seek a new column in which to pivot.
2. $v_{k_4, k_3} \leq 0$. A pivot in column k_3 no longer threatens to violate primal constraint k_4 . We thus compute new ratios using columns k_1 and k_3 of the original primal tableau to determine if we can make a gain on the target constraint k_2 using column k_3 .

	k_5		k_3		k_1	
	-		+		0	k_4
					0	
					0	
	0		-		-	k_2

Figure 2.3: Dual Block in Subproblem (1)

	k_5		$-k_3$	
	-		-	k_4
0	0	0	+	k_2

Figure 2.4: Subproblem (2)

3. We encounter a block in the dual algorithm applied to Subproblem (1). This occurs when a column of Subproblem (1) contains a negative element in row k_4 and a zero in row k_2 , as illustrated in Figure 2.3.

Let k_5 be the column in Subproblem (1) causing the dual block. We now define a further contraction, Subproblem (2), and switch to the primal algorithm. Subproblem (2) consists of all the rows of Subproblem (1), but only those columns of Subproblem (1) with zeros in its target row (k_2). Note that the current target column in Subproblem (1) (k_3) must have a negative element in its "extremal" row (k_2). As a notational convenience we reverse the sign of this element and record this action by labeling the column as $-k_3$. Column $-k_3$ becomes the right-hand side column of Subproblem (2), as shown in Figure 2.4.

The number of columns in Subproblem (2) is reduced by at least one from the number of columns in Subproblem (1). We now apply the primal algorithm to Subproblem (2) until (which may be after zero or more primal pivots):

	k_5		$-k_3$	
	-		-	k_4
			0	
	+		0	k_6
0	0	0	+	k_2

Figure 2.5: Primal Block in Subproblem (2)

1. $v_{k_4, -k_3} \geq 0$, implying that $v_{k_4, k_3} \leq 0$. A pivot in column k_3 no longer threatens to violate primal constraint k_4 . We have thus removed the original primal block, and we proceed by returning to the original primal problem, and computing new ratios using columns k_1 and k_3 .
2. $v_{k_4, k_3} \geq 0$. A pivot in row k_4 no longer threatens to violate dual constraint k_5 , and thus we have removed the dual block in Subproblem (1). We revert to Subproblem (1) and continue with the dual algorithm.
3. We encounter a block in the primal algorithm applied to Subproblem (2). We illustrate this situation in Figure 2.5.

Let k_6 be the row causing the primal block in Subproblem (2). Since the target row in Subproblem (2) (k_2) is identically zero for all but the right-hand side column ($-k_3$), primal unboundedness cannot occur. Also, the entire k_2 row in the original problem is invariant to pivots in this subproblem. We proceed just as we did when faced with this situation in the original primal problem. The current target row (k_4) in the current problem (Subproblem (2)) becomes the extremal function row in a newly defined contraction (Subproblem (3)). All columns of the current problem are retained, but only those rows with zeros in the right-hand side column of the ($-k_3$) current problem are carried forward to the contraction. This again

	k_5		$-k_3$	
			0	
	+		0	k_6
	-		-	k_4

Figure 2.6: Subproblem 3

assures a strict reduction in the number of rows. The new subproblem is shown in Figure 2.6. We then proceed with the dual algorithm, exactly as before.

The construction of the contracting subproblems through the nested sets of zeros in the columns and rows guarantees a monotonic decrease in the sizes of the higher-order subproblems. This ensures the ultimate resolution of degeneracy and gives us a complete, symmetric and finite algorithm for the solution of LP problems.

The blocking resolution scheme given here is a *constructive* algorithm to identify strictly improved solutions. The restricted subproblems ultimately yield a pivot sequence satisfying *all* higher-order criteria. Geometrically, we systematically search the degenerate subspace for an improved representation. This is in sharp contrast to ad hoc "anti-degeneracy" and "anti-cycling" schemes which invoke arbitrary secondary mechanisms not at all related to the geometry or mathematics of the problem at hand, and consequently admit nuisance *degenerate* pivots with no constructive motivation.

I. RELATIONSHIP BETWEEN PRIMAL-DUAL ALGORITHM AND SIMPLEX METHOD

We now depart from the presentation of Graves [1965] to discuss the relationship between the Mutual Primal-Dual Method and the classical Simplex Method. After a brief discussion of the similarities, we will explain our reasons for adapting the Primal-Dual view.

The Simplex Method assumes primal feasibility, and we must identify a starting basic feasible solution. Because this is seldom practical, the usual approach is to formulate and solve a new linear program closely related to the original which has the same optimal solution (assuming one exists) and possesses an easily identified basic feasible solution. This related problem is derived from (PLP) by augmenting it with slack variables, resulting in the following:

$$\begin{aligned}
 (\text{APLP}) \quad & \min_{y,s} : wy + 0s \\
 \text{s.t. : } & Ay + Is = r \quad \} m \\
 & Iy \geq 0 \quad \} n \\
 & Is \geq 0 \quad \} m
 \end{aligned}$$

In the classical Simplex view, a basis is a collection of m linearly independent columns. Let A_B be such a simplex basis which corresponds to a primal feasible solution for (APLP). Suppose we partition the coefficient matrix in a manner determined by the basic slack variables, yielding (after possible row and column interchanges):

$$[A \mid I] = \left[\begin{array}{cc|cc} A_{11} & A_{12} & I_1 & 0 \\ A_{21} & A_{22} & 0 & I_2 \end{array} \right],$$

where

$$A_B = \left[\begin{array}{cc} A_{11} & 0 \\ A_{21} & I_2 \end{array} \right],$$

and the matrix of nonbasic columns is

$$A_{NB} = \left[\begin{array}{cc} I_1 & A_{12} \\ 0 & A_{22} \end{array} \right].$$

The basic variables are $y_B = (y_1, s_2)$ while the nonbasic variables are $y_{NB} = (s_1, y_2)$, and the principal part of the Simplex tableau is $[A_B^{-1} A_{NB}]$.

Borrowing from the perspective of the Primal-Dual algorithm, we may generate the Simplex tableau by partitions:

$$A_B^{-1} = \begin{bmatrix} A_{11} & 0 \\ A_{12} & I_2 \end{bmatrix}^{-1} = \begin{bmatrix} A_{11}^{-1} & 0 \\ -A_{21}A_{11}^{-1} & I_2 \end{bmatrix},$$

and the tableau becomes

$$\begin{aligned} [A_B^{-1} A_{NB}] &= \begin{bmatrix} A_{11}^{-1} & 0 \\ -A_{21}A_{11}^{-1} & I_2 \end{bmatrix} \begin{bmatrix} I_1 & A_{12} \\ 0 & A_{22} \end{bmatrix} \\ &= \begin{bmatrix} A_{11}^{-1} & A_{11}^{-1}A_{12} \\ -A_{21}A_{11}^{-1} & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{bmatrix}, \end{aligned}$$

which is precisely the principal part of the Primal-Dual tableau. This is no surprise when we note that the basis for the Dual Algorithm, T , is exactly A_B , and the Primal and Dual Algorithms share the same tableau. We may interpret the Primal and Dual Algorithms as simply two different perspectives of the same tableau, wherein the Primal Algorithm a pivot is viewed as exchanging primal constraints and in the Dual Algorithm a pivot is viewed as exchanging dual constraints. **The classical Simplex Method may then be interpreted as solving (PLP) using the Dual Algorithm perspective.** That the classical Simplex Method is naturally interpreted as a dual algorithm comes as a surprise to the conventionally trained. However, the consequent mathematical insight is compelling, especially in light of the notational simplification and apparent underlying role of A_{11}^{-1} .

There are several reasons for preferring the Primal-Dual Algorithm to the Simplex Method. From a computational standpoint, because slack variables are carried logically rather than introduced explicitly, we are able to clearly identify the

essential information needed to execute the algorithm. The matrix A_{11}^{-1} plays a key role in the calculation of the tableau, and the entire tableau can be constructed from A_{11}^{-1} and original problem data. Since A_{11}^{-1} is a submatrix of A_B^{-1} , it is smaller and requires fewer arithmetic operations to update than does A_B^{-1} .

A second advantage of the Primal-Dual Algorithm lies in the flexibility it offers for specialization to particular problem classes or structures. Indeed, it is the special structure and simplicity of the nonnegativity constraints that motivate the development of the algorithm in the first place. It is frequently the case that other special structures can be identified in classes of (PLP). Examples of such structures include simple upper bounds, generalized upper bounds, variable upper bounds, pure and generalized network substructures, etc. Such structure may be static in that its nature and dimension remains fixed throughout the solution process, or the structure may be dynamic in which case its precise nature and/or dimension may vary as the problem is solved. Some special structures may be more strongly characterized by their column structure and others by their row structure. The Primal-Dual Algorithm allows one to effectively exploit virtually any such problem structure in a natural manner and greatly simplifies the implementation of such a specialization.

When the linear programming problem appears as a subproblem in a more sophisticated solution setting (for example, in a mixed integer programming problem or a nonlinear programming problem), the row/column symmetry of the Primal-Dual Algorithm is of critical importance in specializing the solution approach. The inherent symmetry of the algorithm permits easy adaptation to branch-and-bound and cutting-plane approaches to mixed integer programming, to column generation settings, as well as to primal and dual decomposition techniques.

We believe the reason for this flexibility offered by the algorithm lies in its more complete mathematical foundation. **There is a natural consistency that arises from the choice of a vector space having the same dimension as the problem variables that is lacking in other approaches.** A natural geometric interpretation of the solution trajectory follows directly from this development. **Incidental issues such as finding an initial basic feasible solution and dealing with degeneracy are resolved constructively in this mathematical framework.** Other approaches resort to unnecessarily complicated tangential efforts.

All the research results reported here can be developed, with some effort, in the framework of the classical Simplex Method. However, we choose to present these results in the manner of their development - the mutual Primal-Dual view presented by Graves.

III. IMPLEMENTATION DESIGN OVERVIEW

A. INTRODUCTION

We seek to demonstrate the efficiency of row factorization (in particular, using dynamic-dimension bases). Accordingly, we will implement the ideas developed here and extensively test them within a commercial-quality optimization system: the X-System [1975] employs the Graves mutual primal-dual algorithm in a variety of large scale optimization applications, including linear, nonlinear, mixed integer and decomposed models. The benchmark test suite is drawn from a wide variety of actual applications, and our goal is to improve the efficiency of an already well-known and highly regarded system.

B. DESIGN CONSIDERATIONS

We want to test our ideas by repeatedly solving many medium- to large-size linear programming problems (i.e., approximately 8,000-10,000 constraints and 15,000-20,000 structural variables). Larger problems are of interest, but for purposes of this research, we are limited to a relatively modest computer budget on an IBM 3033/AP computer under the VM/CMS operating system using VS Fortran 1.4.1. We wish to support the computational enhancements common in commercial mathematical programming systems (e.g., bounded variables, ranged constraints, parametric programming, etc.). We require a primal-dual implementation that offers complete flexibility in determining solution strategy. In addition, each experimental implementation must support all the routine housekeeping of the optimization system (e.g., re-inversion, crash starts, relaxations, restrictions, extrinsic and intrinsic enumeration, etc.). Finally, we seek the capability to identify desired row factorization

structure within the LP model instance, either by communication from the modeler or automatically.

C. DESIGN TEMPLATE

To establish a conceptual framework for the evaluation of our algorithms, it is useful to outline the important aspects of our implementation and identify the crucial steps which most strongly influence performance. Recall from Chapter 2 our basic tableau in partitioned form:

$$\begin{array}{ccc}
 & (j) & (jj) & (jjj) \\
 \begin{array}{l} (i) \\ (ii) \\ (iii) \end{array} & \left(\begin{array}{ccc} A_{11}^{-1} & A_{11}^{-1}A_{12} & A_{11}^{-1}r_1 \\ -A_{21}A_{11}^{-1}A_{12} & A_{22} - A_{21}A_{11}^{-1}A_{12} & g_2 - A_{21}A_{11}^{-1}r_1 \\ -w_1A_{11}^{-1} & w_2 - w_1A_{11}^{-1}A_{12} & -w_1A_{11}^{-1}r_1 \end{array} \right) & (3.1)
 \end{array}$$

where we have made the substitution $y^0 = B^{-1}f$ in the right-hand side column:

$$\begin{aligned}
 y^0 = B^{-1}f & \Rightarrow (y_1^0, y_2^0)^T = \begin{bmatrix} A_{11}^{-1} & A_{11}^{-1}A_{12} \\ 0 & -I \end{bmatrix} \begin{pmatrix} r_1 \\ 0 \end{pmatrix} \\
 & \Rightarrow (y_1^0, y_2^0)^T = (A_{11}^{-1}r_1, 0)^T.
 \end{aligned}$$

The data requirements of the algorithm are:

1. Access to the original problem data;
2. A representation of that part of the current tableau we have chosen to represent implicitly, and;
3. A representation of that part of the current tableau we have chosen to represent explicitly.

We now consider each of these requirements in greater detail.

Primal simplex implementations typically are column-oriented and thus require column-wise access to the problem data. In the primal-dual method, restriction to either column-wise or row-wise access alone exacts a serious computational penalty. Thus, our design allows both column-wise and row-wise access.

The design of an efficient large-scale tableau element generation representation is remarkably complex. Many subtle software engineering and hardware environment issues can have a profound influence on the intricacy and performance of promising designs. The design excursions reported here are inexorably influenced by architecture of the host computer, operating system and implementation language. However, the reported design philosophy has been tempered with experience on many other computers of widely varied designs. The proposed innovations adapt quite well to floating-point pipelines, large cache memories and parallel architectures.

Because of its fundamental role in the construction of the tableau, we maintain an explicit representation of A_{11}^{-1} (we thus refer to A_{11} as the "explicit kernel"). Although any of the various techniques such as LU , LDL^T or QR decomposition or product form inverse (e.g., Golub and Van Loan [1985] or Magnanti [1976]), are suitable for representing A_{11} or A_{11}^{-1} , a difficulty arises in this algorithmic setting. While in the primal Simplex method all updates to the basis take the form of rank one column exchanges, our setting admits more general updates, which include single row exchanges, single row and column exchanges, single row and column deletions, and multiple row exchanges of A_{11}^{-1} (multiple column exchanges of A_{11}). While this does not preclude the use of any particular representation, it adds a level of complexity not usually encountered in more traditional implementations.

We also maintain an explicit representation of the right-hand side column and the bottom (cost) row. Because of the symmetric nature of the mutual primal-dual method, a sensible approach is to allocate a single storage array for both quantities.

which taken together are called the "problem rim". If we generate an explicit representation of the pivot row and pivot column of the tableau at each iteration, then we may update the problem rim array using the simple pivot transformation. By adopting the convention of labeling the nonbasic constraints as rows 1 through m and labeling the basic constraints as rows $(m + 1)$ through $(m + n)$ (assuming our problem instance has m structural constraints and n nonnegativity constraints), the problem rim array is partitioned as follows:

1. The first portion of the array holds values corresponding to the nonbasic constraints in region (i) of the tableau (3.1). Since these are nonnegativity constraints and they are nonbinding (nonbasic), the values in region (i) of the rim array are those of the currently (possibly) nonzero variables.
2. The second portion of the array holds values corresponding to the nonbasic structural constraints in region (ii), and thus the values in the rim array are the current slack or violation in these constraints.
3. The third region of the rim array, beginning at position $m + 1$, corresponds to basic (binding) structural constraints, and thus the values are those of the corresponding (possibly) nonzero dual variables.
4. The fourth region of the rim array corresponds to basic nonnegativity constraints, and thus the values are those of the corresponding (possibly) nonzero dual variables, conventionally called "reduced costs".

The rest of the tableau we represent implicitly, by simply recording in a storage array the current ordering of nonbasic and basic constraints. When additional

information from the tableau (for example, a row or column) is required, we construct it from our representation of A_{11}^{-1} , the current row ordering and the original problem data.

An overview of the solution process is as follows:

1. Identify an initial basic solution. As stated previously, the origin is always such a solution, and thus we may always begin with:

$$B^0 = -I$$

$$D^0 = A,$$

or any other suitable basis.

2. Check for optimality. If there are currently no primal violations (a negative value in the first or second region of the rim array) and there are no dual violations (a positive value in the third or fourth region of the rim array), then the current solution is optimal. Otherwise, we proceed to step 3.
3. Select either a primal or a dual violation, perform a pivot which makes progress towards reducing that violation and return to step 2.

Since we desire to maintain current information in the rim array by means of the pivot transformation updates, we require a representation of the pivot row and pivot column at each iteration. Since we explicitly maintain only A_{11}^{-1} and the problem rim, we see that a key computational step in our implementation is the generation of tableau rows and columns.

Recall from Chapter 2 the principal part of the primal tableau (a symmetric development from the dual perspective is also possible, and is of course equivalent):

$$DP = \begin{array}{cc} & \begin{array}{cc} (j) & (jj) \end{array} \\ \begin{array}{c} (i) \\ (ii) \end{array} & \left(\begin{array}{cc} A_{11}^{-1} & A_{11}^{-1} A_{12} \\ -A_{21} A_{11}^{-1} & A_{22} - A_{21} A_{11}^{-1} A_{12} \end{array} \right) \end{array} \quad (3.2)$$

where $P = -B^{-1}$ is the conjugate row basis,

$$B = \begin{array}{c} (j) \\ (jj) \end{array} \left[\begin{array}{cc} A_{11} & A_{12} \\ 0 & -I \end{array} \right] \quad (3.3)$$

and D contains the nonbasic rows

$$D = \begin{array}{c} (i) \\ (ii) \end{array} \left[\begin{array}{cc} -I & 0 \\ A_{21} & A_{22} \end{array} \right] \quad (3.4)$$

D. SEQUENCING COMPUTATIONS TO EXPLOIT COMMON SUB-EXPRESSIONS IN COLUMN GENERATION

Now consider the generation of column s from (3.2). Rewriting (3.2) in a manner that highlights our intentions:

$$DP = \begin{array}{cc} & \begin{array}{cc} (j) & (jj) \end{array} \\ \begin{array}{c} (i) \\ (ii) \end{array} & \left(\begin{array}{cc} A_{11}^{-1} & A_{11}^{-1} A_{12} \\ -A_{21}(A_{11}^{-1}) & -A_{21}(A_{11}^{-1} A_{12}) + A_{22} \end{array} \right) \end{array} \quad (3.5)$$

By properly sequencing our computations we will exploit the fact that region (ii) of a given column is simply a linear combination of region (i) of the same column.

Assume we want to place the current representation of column s into a work array z , which we partition as $z^T = (z_1, z_2)^T$ to correspond to (3.5). Expressed in terms of an explicit transformation kernel A_{11}^{-1} , we first compute region (i) of column s as:

$$z_1 = (A_{11}^{-1})^s \quad \text{if } s \text{ is in } (j).$$

or

$$z_1 = A_{11}^{-1}(A_{12})^s \quad \text{if } s \text{ is in } (jj).$$

Having done this, we then compute z_2 as:

$$z_2 = -A_{21}z_1 \quad \text{if } s \text{ is in } (j),$$

or

$$z_2 = -A_{21}z_1 + (A_{22})^s \quad \text{if } s \text{ is in } (jj).$$

Assuming an LU representation of A_{11} , we first compute region (i) of column s as:

$$LUz_1 = e^s \quad \text{if column } s \text{ is in } (j),$$

or

$$LUz_1 = (A_{12})^s \quad \text{if column } s \text{ is in } (jj),$$

Having done this, we then compute z_2 as:

$$z_2 = -A_{21}z_1 \quad \text{if column } s \text{ is in } (j),$$

or

$$z_2 = -A_{21}z_1 + (A_{22})^s \quad \text{if column } s \text{ is in } (jj).$$

Then the current representation of column s is available in $z^T = (z_1, z_2)^T$.

E. SEQUENCING COMPUTATIONS TO EXPLOIT COMMON SUB-EXPRESSIONS IN ROW GENERATION

The computation of row t of the tableau proceeds in a similar manner. We now view the principal part of the tableau as:

$$DP = \begin{matrix} & \overbrace{\hspace{1cm}}^{(j)} & \overbrace{\hspace{1cm}}^{(jj)} \\ \begin{matrix} (i) \\ (ii) \end{matrix} & \begin{pmatrix} A_{11}^{-1} & (A_{11}^{-1})A_{12} \\ -A_{21}A_{11}^{-1} & (-A_{21}A_{11}^{-1})A_{12} + A_{22} \end{pmatrix} \end{matrix}.$$

If we want to place the current representation of row t in a work array \tilde{z} partitioned as $\tilde{z} = (\tilde{z}_3, \tilde{z}_4)$, we may first compute region (j) of row t as:

$$\tilde{z}_3 = (A_{11}^{-1})_t \quad \text{if row } t \text{ is in } (i),$$

or

$$\tilde{z}_3 = (-A_{21})_t A_{11}^{-1} \quad \text{if row } t \text{ is in } (ii).$$

We then compute:

$$\tilde{z}_4 = \tilde{z}_3(A_{12}) \quad \text{if row } t \text{ is in } (i),$$

or

$$\tilde{z}_4 = \tilde{z}_3(A_{12}) + (A_{22})_t \quad \text{if row } t \text{ is in } (ii).$$

Alternately using an LU representation of A_{11} ,

$$\tilde{z}_3 LU = e_t \quad \text{if row } t \text{ is in (i),}$$

or

$$\tilde{z}_3 LU = (-A_{21})_t \quad \text{if row } t \text{ is in (ii).}$$

We then compute \tilde{z}_4 as:

$$\tilde{z}_4 = \tilde{z}_3(A_{12}) \quad \text{if row } t \text{ is in (i),}$$

or

$$\tilde{z}_4 = \tilde{z}_3(A_{12}) + (A_{22})_t \quad \text{if row } t \text{ is in (ii),}$$

and the current representation of row t is available in $\tilde{z} = (\tilde{z}_3, \tilde{z}_4)$.

We see that in each case calculations proceed by first using a representation of A_{11} to compute a portion of the row or column and then using this initial computation and original problem data to compute the remaining part. We will discover that our specializations extend this approach by introducing additional tableau partitions which allow this computational strategy to be applied on a larger scale.

As previously mentioned, an important implementation challenge in this algorithmic setting is the dynamic behavior of A_{11} . We see from the primal row basis (3.3) and nonbasic rows (3.4) that the dimension of A_{11} corresponds to the number of basic structural constraints, or, equivalently, to the number of nonbasic nonnegativity constraints (recall that if a nonnegativity constraint is nonbasic and thus

nonbinding, the corresponding variable may possibly be nonzero). Recalling that our primal view of a pivot is as an exchange of constraints between B and D , we see that one of four cases may occur during a pivot:

1. A structural constraint enters the basis B and a structural constraint leaves the basis and enters D . Since the number of basic structural constraints (and the number of nonbasic nonnegativity constraints) remains unchanged, the dimension of A_{11} is unchanged. A pivot of this type involves a row in region (j) of B (3.3) and a row in region (ii) of D (3.4), and thus it corresponds to a pivot coordinate in the location $((ii), (j))$ of the tableau (3.5).
2. A nonnegativity constraint enters the basis and a nonnegativity constraint leaves the basis. Again, the dimension of A_{11} remains unchanged. Since this pivot involves a row in region (jj) of (3.3) and a row in region (i) of (3.4), the corresponding tableau (3.4) pivot coordinate lies in $((i), (jj))$.
3. A structural constraint enters the basis and a nonnegativity constraint leaves the basis, and thus the number of basic structural constraints (equivalently, the number of nonbasic nonnegativity constraints) increases by one. The dimension of A_{11} is increased by one. This corresponds to a pivot coordinate in region $((ii), (jj))$ of the tableau (3.5).
4. A nonnegativity constraint enters the basis and a structural constraint leaves the basis, and thus the dimension of A_{11} is decreased by one. The corresponding pivot coordinate in (3.5) is $((i), (j))$.

We see that we may exert some influence on the behavior of the dimension of A_{11} by our strategy for selecting target violations for primal and dual constraints (i.e., our pricing strategy) and through our tie-breaking rules for choosing pivot

row/columns, and that this dynamism is an inherent feature of our algorithm. We have already seen the fundamental importance of the kernel (A_{11}) in our computations. Thus, a successful implementation must manage this dynamic behavior efficiently and reliably.

To illustrate in a familiar setting the challenge this offers, consider a LU representation of the kernel A_{11} :

$$A_{11} = LU. \quad (3.6)$$

A pivot in tableau coordinate $((i), (jj))$ results in a column exchange in A_{11} . Writing (3.6) in the more convenient form:

$$L^{-1}A_{11} = U. \quad (3.7)$$

When column a^k replaces the p^{th} column of A_{11} to form \bar{A}_{11} , we have

$$L^{-1}\bar{A}_{11} = \begin{array}{|c|c|} \hline & u \\ \hline \alpha & \\ \hline \end{array},$$

where $\alpha = L^{-1}a^k$ has replaced the p^{th} column of U . We now must restore the upper triangular form of U . We would prefer a method which demonstrates strong numerical stability and which also preserves the sparsity of U . Several methods

have been proposed (Bartels and Golub [1969], Forrest and Tomlin [1972], Saunders [1976] and Reid [1982]). Perhaps the most widely used method is that of Saunders.

Assume for the moment that A_{11} has a block (or “bump”) triangular form (we discuss how this is done shortly). Then A_{11} appears as shown in Figure 3.1.

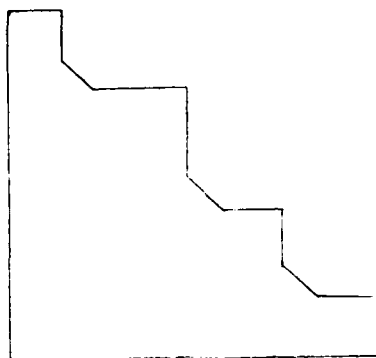


Figure 3.1: Bump Triangular Form of A_{11}

Each bump consists of (possibly) several columns that extend above the main diagonal. These columns are called “spikes”, and the tallest spike within a bump is placed in the right-most column, so that a bump appears as shown in Figure 3.2.

The block triangular form of A_{11} results in a LU decomposition which has the form shown in Figure 3.3.

Saunders exploits the form of these LU factors by maintaining a permutation of the columns and rows of U so that all the spikes appear on the right-hand side without changing their relative ordering, yielding:

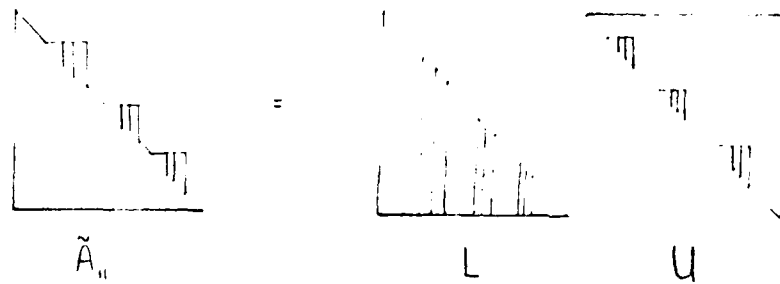


Figure 3.3: LU Decomposition of A_{11}

3. Move the p^{th} row of \hat{U} , \hat{u}_p , to the bottom of \hat{U} , and move all rows in between up one position. Note that this preserves the triangular form of all but the last column of \hat{U} , and further note that row \hat{u}_p has nonzeros only in columns corresponding to R .
4. Using Gaussian elimination with row interchanges, transform \hat{u}_p to a singleton column, thereby restoring the upper triangular form of \hat{U} . Thus, $\hat{U} = E_r \dots E_1 \hat{U}$, where \hat{U} is the final updated form of U and E_r, \dots, E_1 are the elementary transformation matrices corresponding to the Gaussian elimination steps (see, e.g., Murtagh [1981]).
5. Apply these same Gaussian elimination steps to L^{-1} , forming

$$\hat{L}^{-1} = L_r \dots L_1 L^{-1}.$$

Typically, L^{-1} is held in product form and thus these transformations are simply added to the current list of transformation vectors ("eta" vectors).

This method has the virtue that new nonzeros can be created only in the submatrix F of \tilde{U} , and thus R may be carried in peripheral storage. The numerical properties are reported to be quite good.

The computational burden of continuously maintaining A_{11} in block triangular form is excessive, and the usual approach is to refresh the structure as part of the basis re-inversion routine. Between the re-inversions, the structure is left untended. An effective heuristic due to Hellerman and Rarick [1972] is commonly used for this purpose.

A pivot in tableau coordinate $((ii), (j))$ results in a row exchange in A_{11} . Writing (3.6) as

$$A_{11}U^{-1} = L \quad (3.8)$$

When row a_r replaces the q^{th} row of A_{11} to form \bar{A}_{11} , we have

$$\bar{A}_{11}U^{-1} = \begin{bmatrix} \vdots & & \\ \beta & & \\ \vdots & & \\ L & & \\ & & \Delta \end{bmatrix},$$

where row $\beta = a_r U^{-1}$ replaces the q^{th} row of L . The desired structure of L may be restored by methods symmetric to those developed for the column exchange case, again forming :

$$\bar{A}_{11} = \bar{L}\bar{U} .$$

A pivot in tableau coordinate $((ii), (jj))$ causes the dimension of A_{11} to increase by one through the addition of a row and a column. It is convenient to add the new row to the bottom of A_{11} and the new column to its right. If \bar{A}_{11} is the new kernel, then

$$\bar{A}_{11} = \begin{bmatrix} A_{11} & a^k \\ a_r & a_{r,k} \end{bmatrix} , \quad (3.9)$$

where a_r, a^k and $a_{r,k}$ are original matrix coefficients.

The desired updated decomposition is of the form:

$$\begin{bmatrix} A_{11} & a^k \\ a_r & a_{r,k} \end{bmatrix} = \begin{bmatrix} L & 0 \\ l_r & l_{r,r} \end{bmatrix} \begin{bmatrix} U & u^k \\ 0 & u_{k,k} \end{bmatrix} .$$

which requires that

$$Lu^k = a^k$$

$$l_r U = a_r ,$$

and

$$a_{r,k} = l_r u^k + l_{r,r} u_{k,k} .$$

Setting $u_{k,k} = 1$, we have:

$$l_{r,r} = a_{r,k} - l_r u^k .$$

The final pivot coordinate, $((i), (j))$ results in the dimension of A_{11} decreasing by one. If A_{11} is the current transformation kernel and \hat{A}_{11} results after the pivot, without loss of generality we have:

$$A_{11} = \begin{bmatrix} a_{r,k} & a_r \\ a^k & \hat{A}_{11} \end{bmatrix} = \begin{bmatrix} l_{k,k} & 0 \\ l^k & L_1 \end{bmatrix} \begin{bmatrix} u_{r,r} & u_r \\ 0 & U_1 \end{bmatrix}, \quad (3.10)$$

where $(a_{r,k}, a_r)$ is the leaving row, $(a_{r,k}, a^k)^T$ the leaving column and $a_{r,k}$ the pivot element. Using an analysis similar to Rosen [1960], we note that if the square matrix C is nonsingular and is partitioned as:

$$C = \begin{bmatrix} C_1 & C_2 \\ C_3 & C_4 \end{bmatrix},$$

where C_1 is square and nonsingular, C_4 is square and $C_0 = C_4 - C_3 C_1^{-1} C_2$ is nonsingular, then

$$C^{-1} = \begin{bmatrix} C_1^{-1} + C_1^{-1} C_2 C_0^{-1} C_3 C_1^{-1} & -C_1^{-1} C_2 C_0^{-1} \\ -C_0^{-1} C_3 C_1^{-1} & C_0^{-1} \end{bmatrix}.$$

Applying this result to (3.10), we discover that

$$\hat{A}_{11} - a^k (a_{r,k})^{-1} a_r = L_1 U_1.$$

or

$$\hat{A}_{11} = L_1 U_1 + \left(\frac{1}{a_{r,k}} \right) a^k a_r. \quad (3.11)$$

If the term $\left(\frac{1}{a_{r,k}} \right) a^k a_r$ should be the zero matrix, our new decomposition is at hand. Unfortunately, $\left(\frac{1}{a_{r,k}} \right) a^k a_r$ need not be zero, but we may guarantee it to be so by performing a preliminary column exchange update to A_{11} . We replace

column $(a_{r,k}, a^k)^T$ with column $(1, 0)^T$ and compute the updated factors exactly as we did in the column exchange case considered earlier. This results in a modified transformation kernel \tilde{A}_{11} with factors

$$\tilde{A}_{11} = \begin{bmatrix} 1 & a_r \\ 0 & \hat{A}_{11} \end{bmatrix} = \begin{bmatrix} \tilde{l}_{k,k} & 0 \\ \tilde{l}_k & \tilde{L}_1 \end{bmatrix} \begin{bmatrix} \tilde{u}_{r,r} & \tilde{u}_r \\ 0 & \tilde{U}_1 \end{bmatrix}.$$

Then the second term of (3.11) is

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix} 0 \cdot a_r = 0$$

and thus the updated LU factors are at hand:

$$\hat{A}_{11} = \tilde{L}_1 \tilde{U}_1.$$

We now see the advantages and disadvantages of the Graves mutual primal-dual method. It has the advantage that, although the transformation kernel can be as large as m , the number of structural constraints in the problem instance, its dimension is actually equal to the current number of binding structural constraints (or equivalently, to the number of potentially nonzero primal variables). At our initial basic solution, the kernel dimension is zero. If the maximum kernel dimension during the course of the solution trajectory is much smaller than m , we enjoy significant computational advantages in storage, update and computation. For a great many LP model instances, this is precisely the case.

The dynamic nature of the transformation kernel clearly complicates update procedures. While the usual primal Simplex method requires only the column exchange update, Graves' method requires four update cases.

In our implementation, we will seek methods for handling the transformation kernel that enjoy the advantages while mitigating the disadvantages.

It is this general strategy for row and column generation we develop and enhance in our specializations. By identifying a special structure in the problem data, we will introduce additional linear dependencies in the rows and columns of the tableau which will further simplify their generation and will also further reduce the dimension of the transformation kernel.

IV. FACTORIZATION

A. INTRODUCTION

Each of the algorithms we present in this research can be viewed as a specialization of a general approach to large-scale linear programming developed by Graves and McBride [1976], which they call the "factorization approach". It is based on distinguishing special rows and columns in a way that allows large parts of the basic tableau to be represented implicitly, to be generated easily from the remaining explicit parts only when actually required by the algorithm.

Although sometimes used interchangeably in the literature, we recognize a distinction between partitioning methods and factorization methods. In the former, a formal distinction is made between substructures which appear in the model instance, usually constraints or variables. An approach for solving the problem is then developed which exploits the substructures. For example, Dantzig-Wolfe decomposition is such an approach which partitions constraints, while in Benders decomposition the variables are partitioned. Such an approach may be applied statically, in which case the desired partitions are identified at the start of the solution process and remain fixed throughout, or it may be dynamic, in which case the partitioning may be adjusted as the solution proceeds.

In contrast, we consider factorization methods to be those in which a formal distinction is made between substructures in the LP basis (and thus in the basic tableau). The algorithm is then specialized to exploit this special substructure. Thus, we view the GUB algorithm of Dantzig and Van Slyke [1967] and the multicommodity network flow algorithm of Hartman and Lasdon [1970] as examples

of factorization algorithms. Factorization methods may also be static or dynamic, depending on whether the dimension of the factored substructures in the LP basis may vary during the solution process.

Our research is concerned with dynamic row factorization algorithms. We develop the general dynamic factorization approach here and discuss the important aspects of the algorithm. In subsequent chapters, we specialize this general approach to each of several special LP row structures. This general development closely parallels that of Graves and McBride [1976].

B. THE FACTORED TABLEAU

The problem to be considered is:

$$\begin{array}{ll}
 \text{(FLP)} \quad \min : & wy \\
 \text{s.t. :} & Ey \leq r \quad \} \text{explicit constraints} \\
 & Fy \leq b \quad \} \text{factored constraints} \\
 & -Iy \leq 0 \quad \} \text{nonnegativity constraints} \quad ,
 \end{array}$$

where y is a n vector of decision variables, w is a n vector of cost coefficients, E is an m by n matrix of constraint coefficients for "explicit" constraints with right-hand side m vector r , F is a p by n matrix of constraint coefficients for "factored" constraints with right-hand side p vector b , and $-I$ is the negative of the n by n identity matrix. In this general development, we refer to the F -type constraints as "factored" only to distinguish them from the "explicit" E -type constraints, and assume nothing about their structure. Not until our specializations in later chapters will we impose special structure on F , and the structures we will consider permit the representation of the F type constraints without the inversion of a matrix. We will see that this approach is centered around handling the part of the basis corresponding to the E -type constraints explicitly while factoring the portion of the

basis corresponding to the F -type constraints. The notation is chosen to suggest this idea.

As we saw in Chapter 2, in the mutual primal-dual method the primal algorithm and the dual algorithm share the same tableau, and thus the tableau for (FLP) may be derived from either perspective. We present only the derivation from the primal viewpoint here.

Recall that a basis for the primal algorithm consists of n linearly independent rows from the constraint coefficient matrix when it is assumed to include both structural (explicit and factored) and nonnegativity constraints. Assume that the current row basis consists of l rows from E , k rows from F and $(n - k - l)$ rows from $-I$. Using the notation of Chapter 2 temporarily, we have:

$$B = \left(\begin{array}{cc} \overbrace{\quad}^{l+k} & \overbrace{\quad}^{n-l-k} \\ A_{11} & A_{12} \\ 0 & -I \end{array} \right) \begin{array}{l} \} l+k \\ \} n-l-k \end{array} ,$$

where $[A_{11} \ A_{12}]$ includes all basic structural rows, from both E and F .

We will ultimately be interested in isolating the effect of each type of structural constraint algebraically in the factored tableau, and thus we require greater resolution in our factored basis. Introducing obvious notation, we have:

$$[A_{11} \ A_{12}] = \left(\begin{array}{ccc} \overbrace{\quad}^k & \overbrace{\quad}^l & \overbrace{\quad}^{n-l-k} \\ E_{11} & E_{12} & E_{13} \\ F_{11} & F_{12} & F_{13} \end{array} \right) \begin{array}{l} \} l \\ \} k \end{array} ,$$

where

$$[A_{11}] = \begin{bmatrix} E_{11} & E_{12} \\ F_{11} & F_{12} \end{bmatrix} .$$

From the development in Chapter 2, we recognize A_{11} as the explicit kernel, and thus A_{11}^{-1} exists. It then follows that it is always possible to identify among the columns of $[F_{11} \ F_{12}]$ a nonsingular submatrix F_{11} of dimension k , since otherwise the rank of $[F_{11} \ F_{12}]$ is at most $(k - 1)$ and thus the rank of B is at most $(n - 1)$. We will later see that one of the important implementation challenges is the task of efficiently managing the structure and nonsingularity of F_{11} .

The full factored row basis is then:

$$B = \begin{matrix} & \overbrace{\quad}^k & \overbrace{\quad}^l & \overbrace{\quad}^{n-l-k} \\ \begin{matrix} (j) \\ (jj) \\ (jjj) \end{matrix} & \begin{pmatrix} E_{11} & E_{12} & E_{13} \\ F_{11} & F_{12} & F_{13} \\ 0 & 0 & -I \end{pmatrix} & \begin{matrix} \} l \\ \} k \\ \} n-l-k \end{matrix} \end{matrix} \quad (4.1)$$

Introducing the notation:

$$\begin{aligned} \hat{A}_{11} &\triangleq E_{12} - E_{11}F_{11}^{-1}F_{12} \\ \hat{A}_{12} &\triangleq E_{13} - E_{11}F_{11}^{-1}F_{13} \end{aligned} ,$$

its inverse is:

$$B^{-1} = \begin{bmatrix} -F_{11}^{-1}F_{12}\hat{A}_{11}^{-1} & (I + F_{11}^{-1}F_{12}\hat{A}_{11}^{-1}E_{11})F_{11}^{-1} & F_{11}^{-1}(F_{13} - F_{12}\hat{A}_{11}^{-1}\hat{A}_{12}) \\ \hat{A}_{11}^{-1} & -\hat{A}_{11}^{-1}E_{11}F_{11}^{-1} & \hat{A}_{11}^{-1}\hat{A}_{12} \\ 0 & 0 & -I \end{bmatrix} .$$

Grouping the coefficients of the nonbasic constraints and applying the same column ordering yields:

$$D = \begin{matrix} & \overbrace{\quad k \quad} & \overbrace{\quad l \quad} & \overbrace{\quad n-l-k \quad} \\ \begin{matrix} (i) \\ (ii) \\ (iii) \\ (iv) \end{matrix} & \begin{pmatrix} -I & 0 & 0 \\ F_{21} & F_{22} & F_{23} \\ 0 & -I & 0 \\ E_{21} & E_{22} & E_{23} \end{pmatrix} & \begin{matrix} \} k \\ \} p-k \\ \} l \\ \} m-l \end{matrix} \end{matrix} \quad (4.2)$$

We will explain the ordering of the rows of D shortly.

The principal part of the factored tableau is DP , where $P = -B^{-1}$ is the conjugate row basis. Introducing the additional notation:

$$\begin{aligned} \hat{F}_{22} &\triangleq F_{22} - F_{21}F_{11}^{-1}F_{12} \\ \hat{F}_{23} &\triangleq F_{23} - F_{21}F_{11}^{-1}F_{13} \\ \hat{A}_{21} &\triangleq E_{22} - E_{21}F_{11}^{-1}F_{12} \\ \hat{A}_{22} &\triangleq E_{23} - E_{21}F_{11}^{-1}F_{13} \end{aligned}$$

the principal part of the factored tableau is:

$$DP = \begin{matrix} & \overbrace{\quad (i) \quad} & \overbrace{\quad (ii) \quad} & \overbrace{\quad (iii) \quad} \\ \begin{matrix} (i) \\ (ii) \\ (iii) \\ (iv) \end{matrix} & \begin{pmatrix} -F_{11}^{-1}F_{12}\hat{A}_{11}^{-1} & (I + F_{11}^{-1}F_{12}\hat{A}_{11}^{-1}E_{11})F_{11}^{-1} & F_{11}^{-1}(F_{13} - F_{12}\hat{A}_{11}^{-1}\hat{A}_{12}) \\ -\hat{F}_{22}\hat{A}_{11}^{-1} & (\hat{F}_{22}\hat{A}_{11}^{-1}E_{11} - F_{21})F_{11}^{-1} & \hat{F}_{23} - \hat{F}_{22}\hat{A}_{11}^{-1}\hat{A}_{12} \\ \hat{A}_{11}^{-1} & -\hat{A}_{11}^{-1}E_{11}F_{11}^{-1} & \hat{A}_{11}^{-1}\hat{A}_{12} \\ -\hat{A}_{21}\hat{A}_{11}^{-1} & (\hat{A}_{21}\hat{A}_{11}^{-1}E_{11} - E_{21})F_{11}^{-1} & \hat{A}_{22} - \hat{A}_{21}\hat{A}_{11}^{-1}\hat{A}_{12} \end{pmatrix} \end{matrix} \quad (4.3)$$

Partitioning $w = (w_1, w_2, w_3, w_4)$, $r^T = (r_1, r_2)^T$ and $b^T = (b_1, b_2)^T$ and introducing the notation:

$$\hat{w}_1 \triangleq w_1 - w_1 F_{11}^{-1} F_{12}$$

$$\begin{aligned}
\hat{w}_3 &\triangleq w_3 - w_2 F_{11}^{-1} F_{13} \\
\hat{b}_2 &\triangleq b_2 - F_{21} F_{11}^{-1} b_1 \\
\hat{r}_1 &\triangleq r_1 - E_{11} F_{11}^{-1} b_1 \\
\hat{r}_2 &\triangleq r_2 - E_{21} F_{11}^{-1} b_1,
\end{aligned}$$

the complete factored tableau is:

$$\begin{bmatrix}
-F_{11}^{-1} F_{12} \hat{A}_{11}^{-1} & (I + F_{11}^{-1} F_{12} \hat{A}_{11}^{-1} E_{11}) F_{11}^{-1} & F_{11}^{-1} (F_{13} - F_{12} \hat{A}_{11}^{-1} \hat{A}_{12}) & F_{11}^{-1} (b_1 - F_{12} \hat{A}_{11}^{-1} \hat{r}_1) \\
-\hat{F}_{22} \hat{A}_{11}^{-1} & (\hat{F}_{22} \hat{A}_{11}^{-1} E_{11} - F_{21}) F_{11}^{-1} & \hat{F}_{23} - \hat{F}_{22} \hat{A}_{11}^{-1} \hat{A}_{12} & \hat{b}_2 - \hat{F}_{22} \hat{A}_{11}^{-1} \hat{r}_1 \\
\hat{A}_{11}^{-1} & -\hat{A}_{11}^{-1} E_{11} F_{11}^{-1} & \hat{A}_{11}^{-1} \hat{A}_{12} & \hat{A}_{11}^{-1} \hat{r}_1 \\
-\hat{A}_{21} \hat{A}_{11}^{-1} & (\hat{A}_{21} \hat{A}_{11}^{-1} E_{11} - E_{21}) F_{11}^{-1} & \hat{A}_{22} - \hat{A}_{21} \hat{A}_{11}^{-1} \hat{A}_{12} & \hat{r}_2 - \hat{A}_{21} \hat{A}_{11}^{-1} \hat{r}_1 \\
-w_2 \hat{A}_{11}^{-1} & (\hat{w}_2 \hat{A}_{11}^{-1} E_{11} - w_1) F_{11}^{-1} & \hat{w}_3 - \hat{w}_2 \hat{A}_{11}^{-1} \hat{A}_{12} & w_1 F_{11}^{-1} b_1 + \hat{w}_2 \hat{A}_{11}^{-1} \hat{r}_1
\end{bmatrix} \quad (4.4)$$

We see from (4.4) that with knowledge of the current factorization, we can construct the entire tableau from F_{11}^{-1} , \hat{A}_{11}^{-1} and the original problem data. The dimension of F_{11}^{-1} is equal to the number of F -type constraints that are currently basic, and thus can be at most p . The dimension of \hat{A}_{11}^{-1} is equal to the number of E -type constraints that are currently basic. Hence, its dimension cannot exceed m . We call \hat{A}_{11}^{-1} the explicit transformation kernel and F_{11}^{-1} the factored transformation kernel.

We have seen in Chapter 3 that the origin is always a convenient initial basic solution for the mutual primal-dual method, and the same is true for the factorization approach. An initial basic solution is always:

$$B^0 = [-I]$$

$$D^0 = \begin{bmatrix} F \\ E \end{bmatrix}.$$

Starting from this solution, we may view the algorithm as progressing by exchanging rows of F and E from D with nonnegativity constraints from B . We will find it useful to associate with each F -type constraint in B a unique nonnegativity constraint in D . Borrowing the terminology of Dantzig and Van Slyke [1967], we will call each such unique nonnegativity constraint (and its corresponding variable) the "key" variable for the associated F -type constraint. The algebraic structure arising from this basic F -type constraint/nonbasic "key" variable association allows us to represent parts of the tableau implicitly rather than explicitly. To distinguish these "key" variables in D from others, we place them in region (i) of D . This is the reason for the row ordering mentioned earlier.

C. BENEFITS OF FACTORIZATION

Graves and McBride [1976] report three principal benefits of the factorization approach:

1. Good starting bases;
2. Reduction in memory requirements;
3. Reduction in work per pivot.

Although the origin is always available as an initial basic solution, Graves and McBride [1976] suggest that a better one can frequently be found with only small

computational effort using this approach. They form the Lagrangian Dual of (FLP) with respect to the E -type constraints, we obtain:

$$\begin{aligned} \text{(LFLP)} \quad \min_y : \quad & wy + \lambda(Ey - r) \\ \text{s.t. :} \quad & Fy \leq b \\ & -Iy \leq 0 \end{aligned}$$

where $\lambda \geq 0$ is a m vector of dual variables. If λ equals λ^* , the optimal dual variables for (FLP), then an optimal solution of (FLP) is among the optimal solutions of (LFLP). They speculate that if $\hat{\lambda}$ is a "good" estimate of λ^* , then using $\hat{\lambda}$ in (LFLP) and solving for y should yield a "good" starting point \hat{y} for (FLP). If no such estimate $\hat{\lambda}$ is available, then $\lambda = 0$ may be used. Depending upon the structure of the F -type constraints, the resulting problem may either be much easier to solve than (FLP) and thus may be solved optimally, or heuristics may be used to find a computationally inexpensive "good" solution. All factorization algorithms can be started by solving (LFLP) first.

Standard simplex techniques require a basis of dimension $(m + p)$ to solve (FLP). The factorization approach requires the explicit transformation kernel \hat{A}_{11}^{-1} , whose dimension is at most m , and the factored kernel F_{11} , whose dimension is at most p . Thus, it is clear that we expect a considerable reduction in memory requirements using the factorization approach. When we specialize the approach for models in which the F -type constraints have special structure (e.g., GUB, pure network or generalized network), we will see that memory requirements can be further reduced.

While it is difficult to measure the computational effort per pivot when product form inverse or basis decomposition techniques are used, there are indications that the factorization approach can yield substantial benefits. Both analytic (McBride

[1972]) and empirical (Tomlin [1972]) studies indicate that when the F -type constraints are GUB constraints (we will cover this in detail in Chapter 6), the factorization approach results in per-pivot computational benefits when the GUB constraints comprise at least 30% of the total number of structural constraints. With some simplifying assumptions, a similar analysis can be developed for the case where the F -type constraints are pure network rows.

We will consider each of three different factored row structures in this work. The complexity of the factored row structure determines the computational difficulty of representing and updating the factored kernel F_{11} . The simplest row structure we consider, in which the factored rows are generalized upper bounds, allows F_{11} to be interpreted as a simple permutation matrix. The second factorization, in which the F -type constraints are pure network rows, is a more complex structure which subsumes generalized upper bounds as a special case. In this case, F_{11} may be interpreted as representing a partial ordering defined over a subset of the rows of F . The final factorization, in which the rows of F are generalized network rows, is the most general we consider in this work and subsumes both the other factorizations. Here, F_{11} may be interpreted as a linear transformation in which a near partial ordering exists among a subset of the rows of F .

V. GENERAL IMPLEMENTATION TOOLS

A. INTRODUCTION

We now provide an overview of our implementation of the factorization method. We describe the representation and update of the three important algorithm components: the basic tableau, the factored kernel and the explicit transformation kernel. The fundamental update steps are described in terms of functions which operate on each of the algorithm components. Where the details of the update actions are common to all three factorizations treated in this research, the details are presented in this chapter; otherwise, they appear in subsequent chapters.

B. THE FACTORED TABLEAU

As before, we are interested in the problem.

$$\begin{array}{ll} \min_y : & wy \\ \text{s.t.} : & \begin{array}{l} Ey \leq r \} \text{ explicit constraints} \\ Fy \leq b \} \text{ factored constraints} \\ -Iy \leq 0 \} \text{ nonnegativity constraints} \end{array} \end{array}$$

Recall the algebraic representation of an arbitrary primal row basis:

$$B = \begin{pmatrix} (j) \\ (jj) \\ (jjj) \end{pmatrix} \begin{pmatrix} E_{11} & E_{12} & E_{13} \\ F_{11} & F_{12} & F_{13} \\ 0 & 0 & -I \end{pmatrix} \quad (5.1)$$

with the corresponding nonbasic rows:

$$D = \begin{matrix} (i) \\ (ii) \\ (iii) \\ (iv) \end{matrix} \begin{pmatrix} -I & 0 & 0 \\ F_{21} & F_{22} & F_{23} \\ 0 & -I & 0 \\ E_{21} & E_{22} & E_{23} \end{pmatrix} \quad (5.2)$$

The conjugate row basis inverse is:

$$P = -B^{-1} = \begin{pmatrix} F_{11}^{-1}F_{12}\hat{A}_{11}^{-1} & -F_{11}^{-1} - F_{11}^{-1}F_{12}\hat{A}_{11}^{-1}E_{11}F_{11}^{-1} & -F_{11}^{-1}F_{13} + F_{11}^{-1}F_{12}\hat{A}_{11}^{-1}(E_{13} - E_{11}F_{11}^{-1}F_{13}) \\ -\hat{A}_{11}^{-1} & \hat{A}_{11}^{-1}E_{11}F_{11}^{-1} & -\hat{A}_{11}^{-1}(E_{13} - E_{11}F_{11}^{-1}F_{13}) \\ 0 & 0 & I \end{pmatrix} \quad (5.3)$$

and the principal part of the factored tableau is:

$$\begin{matrix} (i) \\ (ii) \\ (iii) \\ (iv) \end{matrix} \begin{pmatrix} -F_{11}^{-1}F_{12}\hat{A}_{11}^{-1} & F_{11}^{-1} + F_{11}^{-1}F_{12}\hat{A}_{11}^{-1}E_{11}F_{11}^{-1} & F_{11}^{-1}F_{13} - F_{11}^{-1}F_{12}\hat{A}_{11}^{-1}(E_{13} - E_{11}F_{11}^{-1}F_{13}) \\ -F_{22}\hat{A}_{11}^{-1} + F_{21}F_{11}^{-1}F_{12}\hat{A}_{11}^{-1} & F_{22}\hat{A}_{11}^{-1}E_{11}F_{11}^{-1} - F_{21}F_{11}^{-1} & F_{23} - F_{21}F_{11}^{-1}F_{13} + F_{21}F_{11}^{-1}F_{12}\hat{A}_{11}^{-1}(E_{13} - E_{11}F_{11}^{-1}F_{13}) \\ & -F_{21}F_{11}^{-1}F_{12}\hat{A}_{11}^{-1}E_{11}F_{11}^{-1} & -F_{21}\hat{A}_{11}^{-1}(E_{13} - E_{11}F_{11}^{-1}F_{13}) \\ \hat{A}_{11}^{-1} & -\hat{A}_{11}^{-1}E_{11}F_{11}^{-1} & \hat{A}_{11}^{-1}(E_{13} - E_{11}F_{11}^{-1}F_{13}) \\ -E_{11}\hat{A}_{11}^{-1} + E_{12}F_{11}^{-1}F_{12}\hat{A}_{11}^{-1} & E_{12}\hat{A}_{11}^{-1}E_{11}F_{11}^{-1} - E_{12}F_{11}^{-1} & E_{13} + E_{12}F_{11}^{-1}F_{12}\hat{A}_{11}^{-1}(E_{13} - E_{11}F_{11}^{-1}F_{13}) \\ & -E_{12}F_{11}^{-1}F_{12}\hat{A}_{11}^{-1}E_{11}F_{11}^{-1} & -E_{12}\hat{A}_{11}^{-1}(E_{13} - E_{11}F_{11}^{-1}F_{13}) \end{pmatrix} \quad (5.4)$$

The entire principal part of the factored tableau can be constructed with knowledge of the row and column ordering, F_{11}^{-1} (or F_{11}) and \tilde{A}_{11}^{-1} . Our discussion of the implementation centers around the representation and update of these three components.

We restrict our attention to the principal part of the tableau because our implementation maintains a current representation of the right-hand side column and bottom (cost) row (collectively referred to as the "problem rim") at all times. These quantities are always immediately available and need not be computed upon demand. This is in contrast to (5.4), where only \tilde{A}_{11}^{-1} and F_{11} are always available and any other quantity of interest (for example, a complete row or column) must be computed.

C. ALGORITHM OVERVIEW

In broad terms, the algorithm proceeds as follows:

1. Establish an initial basic solution.
2. Stop if the current solution is optimal. Optimality exists when the right-hand side column is nonnegative and the bottom (cost) row is nonpositive.
3. Based on a pricing scheme and a ratio test, select and generate a row and column of (5.4).
4. Stop if this row and column reveal infeasibility or unboundedness.
5. Transform the problem rim, the tableau representation, F_{11} and \tilde{A}_{11}^{-1} , and return to step 2.

After developing the data structures and necessary update operations, we will expand upon this overview.

D. IMPLEMENTATION CONVENTIONS

We will first discuss the implementation of each of the three algorithm components individually, and then provide a detailed presentation of the complete algorithm. Our implementation is coded in the FORTRAN language, and thus the following discussion is colored by the character of that language.

1. The Tableau

The mutual primal-dual method fosters a symmetric view of the linear programming problem and the solution process. This symmetry is reflected in the indexing conventions required in our implementation. Assuming now that (5.1) consists of m structural constraints (both factored and explicit) and n structural variables, we require the structural constraints to be indexed from 1 to m , and the variables from $(m + 1)$ to $(m + n)$. While our statement of the problem specifies the nonnegativity constraints explicitly, we will of course deal with them implicitly. Note, however, that we have two equivalent interpretations of the indices $(m + 1)$ through $(m + n)$. We can view them as the indices of the structural problem variables or as the indices of the nonnegativity constraints corresponding to each of the structural variables. We will use both interpretations in our development. Observe also that we require no explicit representation of logical (slack, surplus or artificial) variables.

The indexing of the original problem data exists independently of the solution process and is thus referred to as the "extrinsic" indexing. The factoring of the constraints to form B and D represents a second indexing of the problem which defines the current solution, independent of the extrinsic indexing. We refer to this indexing system as "intrinsic", and it partially defines the current representation of B and D . Our convention for intrinsic indexing has the rows of D labeled from 1 to m and the rows of B labeled from $(m + 1)$ to $(m + n)$.

The original problem data is stored in super-sparse form, (in super-sparse form, each real value is stored once, and each nonzero coefficient is represented by a row index, a column index, and a pointer to a real value) with unique nonzero real values stored once in an array of type DOUBLE PRECISION, and referenced by the FORTRAN equivalent of a pointer from each constraint matrix coefficient. These pointers to the real values are singly-linked by both row (with associated column indices) and by column (with associated row indices). This allows symmetric row-wise and column-wise access to the data to be performed efficiently, which is an important virtue in a primal-dual design.

To specify the structure of (5.4) we require a representation of the current mapping between the extrinsic indexing of the problem data and the intrinsic indexing of the current tableau. We represent this mapping by two arrays of type INTEGER, each dimensioned from 1 to $(m + n)$, which are used as pointers. We denote these arrays MINT() and MEXT(). MINT() represents the mapping from intrinsic to extrinsic indices, and MEXT() represents the inverse mapping. Thus, if IINT is an intrinsic index of a row or column of (5.4) and IEXT is the extrinsic index of the constraint or variable currently mapped to position IINT of the tableau, then:

$$\begin{aligned} \text{MINT(IINT)} &= \text{IEXT} \\ \text{MEXT(IEXT)} &= \text{IINT} \\ \text{MINT(MEXT(IEXT))} &= \text{IEXT} \\ \text{MEXT(MINT(IINT))} &= \text{IINT} \end{aligned}$$

To complete the representation of (5.4) we require knowledge of the factorization, which is represented algebraically by regions (n) , (ni) , (nii) , etc. We

handle this with the use of several variables of type INTEGER which are used as pointers to record the ending intrinsic index of the various tableau regions. Table (5.1) lists the regions and the associated pointer names.

TABLE 5.1: Indices of Tableau Regions

REGION	BEGINNING INDEX	ENDING INDEX
(i)	1	MKC
(ii)	MKC + 1	MFR
(iii)	MFR + 1	MXR
(iv)	MXR + 1	M
(j)	M + 1	NXR
(jj)	NXR + 1	NFR
(jjj)	NFR + 1	M + N

The arrays MINT() and MEXT() and the pointers MKC, MFR, MXR, NXR and NFR comprise the data structures necessary to represent the principal part of the tableau. To complete the representation of the tableau we require a representation of the right-hand side column, the bottom row and the current value of the extremal function. The right-hand side column and bottom row are represented by an array of type DOUBLE PRECISION, dimensioned from 1 to $(m+n)$, referred to as RIM(). Positions 1 through m correspond to the right-hand side column and positions $(m+1)$ through $(m+m)$ correspond to the bottom row. Note that these are intrinsic indices. The current value of the extremal function is held in a scalar variable of type DOUBLE PRECISION called OPT.

We now discuss the operations necessary to maintain and update the tableau representation. To recognize a feasible solution we simply scan the RIM() array. When the values in positions 1 through m are nonnegative and those in positions $(m+1)$ through $(m+n)$ are nonpositive, the current solution is optimal.

Violations of either sign discipline may serve as potential primal algorithm target rows or dual algorithm target columns, respectively.

Having selected a target row or column, we must generate the corresponding row or column of the tableau, compute ratios with respect to the right-hand side column or bottom row, and thus identify the index of a corresponding column or row. We then generate that column or row of the tableau. Because the details of row and column generation differ according to the factorization, we defer their discussion until the appropriate chapter. To represent the update operations, however, we define the abstract functions **Generate_Row**(IPRI) and **Generate_Column**(IPCI). These functions accept as arguments the intrinsic row (IPRI) or column (IPCI) index of the current tableau and return the current representation of that row or column. We require a working array of type DOUBLE PRECISION to hold these current representations. We define ROWCOL() to be such an array, dimensioned from 1 to $(m + n)$, where positions 1 through m correspond to column IPCI of (5.4) and positions $(m + 1)$ through $(m + n)$ correspond to row IPRI. ROWCOL() is indexed intrinsically in conformance with our convention for (5.4).

Recall that our interpretation of a pivot from the primal algorithm viewpoint is as an exchange of rows between B and D . The row indices of D correspond to the row indices of (5.4), while the row indices of B correspond to column indices of (5.4). Thus, a pivot requiring the exchange of a row of B and a row of D requires the exchange of a row index of (5.4) with a column index of (5.4). Using MINT() and MEXT(), such an exchange requires just a few assignment statements. For example, if the pivot row IPRI corresponds to extrinsic index EPRI and the pivot column IPCI corresponds to extrinsic index EPCI, then prior to a pivot we have:

$$\text{MINT}(\text{IPRI}) = \text{EPRI}$$

$$\text{MEXT}(\text{EPRI}) = \text{IPRI}$$

$$\text{MINT}(\text{IPCI}) = \text{EPCI}$$

$$\text{MEXT}(\text{EPCI}) = \text{IPCI}$$

and after the pivotal exchange we have:

$$\text{MINT}(\text{IPRI}) = \text{EPCI}$$

$$\text{MEXT}(\text{EPCI}) = \text{IPRI}$$

$$\text{MINT}(\text{IPCI}) = \text{EPRI}$$

$$\text{MEXT}(\text{EPRI}) = \text{IPCI}$$

We call this type of index exchange a "primary exchange". For abstraction purposes we define a function **Primary_Index_Exchange** (IPRI,IPCI) which, given the current tableau representation and the intrinsic pivot row index IPRI and the intrinsic pivot column index IPCI, performs the appropriate update of MINT() and MEXT(). Every pivot requires a primary exchange.

In addition to the partitioning of constraints into B and D , we are required to maintain the factorizations ((i), (ii), etc.) within each. Having performed the primary exchange, it is possible that additional exchanges within B and/or D are required to restore the proper factorization. For example, if the primary exchange removes an F -type constraint from D (region (ii)) and places it in region (j) of B , an additional exchange is necessary to move the row from region (j) to region (jj). We refer to such actions as "secondary exchanges". Secondary exchanges in D correspond to row exchanges in (5.4) and secondary exchanges in B to column exchanges in (5.4). Thus, we define the functions **Row_Index_Exchange**(IRI1,IRI2)

and **Column_Index_Exchange**(ICI1,ICI2) which exchange intrinsic rows IRI1 and IRI2 or intrinsic columns ICI1 and ICI2, respectively. Since the indexing of the **RIM()** array corresponds to the tableau indexing, we assume that any secondary exchanges performed on the tableau are also performed on **RIM()**.

Finally, our factorization requires that the factored kernel, F_{11} , remain nonsingular. It is possible that the primary and secondary exchanges will destroy the nonsingularity of F_{11} . When this occurs, additional row exchanges between regions (i) and (iii) of D will be necessary to restore the required nonsingularity of F_{11} . We call such exchanges "tertiary exchanges". Again, we assume that tertiary exchanges performed on the tableau are also performed on **RIM()**.

The factorizations of B and D are dynamic in nature, meaning that a particular pivot may cause the dimension of one or more regions of B and/or D to increase or decrease by one row. Such dimension changes are handled by simply adjusting the values of the factorization pointers. We define the functions **Increment(XXX)** and **Decrement(XXX)**, where "XXX" is MKC, MFR, MXR, NXR or NFR, to increment or decrement, respectively, the appropriate pointer value.

In summary, the tableau data structures are:

MINT()

MEXT()

RIM()

ROWCOL()

MKC, MFR, MXR, NFR, NXR.

and the necessary update operations are:

Generate_Row(*IPRI*)

Generate_Column(*IPCI*)

Primary_Index_Exchange(*IPRI*, *IPCI*)

Row_Index_Exchange(*IRI1*, *IRI2*)

Column_Index_Exchange(*ICI1*, *ICI2*)

Increment(*XXX*), and

Decrement(*XXX*).

2. The Explicit Transformation Kernel

As can be seen from (5.4), the rows of \tilde{A}_{11}^{-1} correspond to region (iii) of the tableau (nonbasic nonkey variables) and the columns to region (j) (basic explicit constraints). The dimension of \tilde{A}_{11}^{-1} may vary as the algorithm progresses, and the mechanism for these changes is the addition and deletion of rows and columns. We thus require data structures that permit the efficient insertion, deletion and exchange of rows and columns of \tilde{A}_{11}^{-1} . Further, the number of nonzero elements in \tilde{A}_{11}^{-1} varies from pivot to pivot, independently of dimension changes. We store these nonzero elements in a stack, referring to them via pointers which are stored as an orthogonally linked list, doubly linked by row and doubly linked by column. This allows the efficient update of \tilde{A}_{11}^{-1} and accommodates the dynamics, at some expense in storage. Contrary to the tableau representation, we maintain the explicit transformation kernel representation indexed extrinsically rather than intrinsically. This allows permutations within region (j) and/or within region (iii) of the tableau

without requiring corresponding permutations in the columns and/or rows of the \tilde{A}_{11}^{-1} representation.

We again define a suite of functions which operate on our representation of \tilde{A}_{11}^{-1} and describe its update. **Add_Row(IRE)** and **Add_Column(IJE)** append extrinsic row IRE and extrinsic column IJE, respectively, to our representation of \tilde{A}_{11}^{-1} . We will ensure that the current representation of each is in work array ROWCOL() prior to executing these functions. Columns will always be appended to the right-hand side of \tilde{A}_{11}^{-1} and rows will be appended to the bottom. **Delete_Row(IRE)** and **Delete_Column(IJE)** delete extrinsic row IRE and IJE, respectively, from the representation of \tilde{A}_{11}^{-1} . **Replace_Row(IRE1,IRE2)** causes the overlay of extrinsic row IRE1 by extrinsic row IRE2. The representation of IRF will have been previously placed in ROWCOL(). Finally, **Update_Explicit_Transformation_Kernel** performs a pivot update of the representation of \tilde{A}_{11}^{-1} , using the current representation of the pivot row and pivot column in ROWCOL().

3. The Factored Kernel

The data structures and update actions for the factored kernel vary according to the factorization and their discussion will be deferred. Here we define the necessary abstract update functions.

Is_Factored_Kernel_Singular tests the current representation of F_{11} for singularity and returns the appropriate Boolean value. It will be used in certain pivot cases to determine which of several courses of action should be taken.

All tertiary exchanges take the form of row exchanges between regions (i) and (iii) of (5.4). Note that region (i) consists of the nonnegativity constraints of key variables. By our alternate interpretation of the nonnegativity constraint indices we may consider these to be key variable indices. Since the columns of F_{11} are key variables, we interpret region (i) as containing the indices of the columns

of F_{11} . An exchange of rows between (i) and (iii) of (5.4) is then interpreted as an exchange of columns of F_{11} .

Before such a column exchange for F_{11} (and thus a row exchange in (5.4)) can be made, we must frequently identify the index of the column to be removed from F_{11} (which is an index in region (i) of (5.4)) and the index of the column which will replace it (which is an index in region (iii) of (5.4)). We thus define **Find_Column_to_Remove(IOUT)**, which selects from among the indices of region (i) of (5.4) the intrinsic index IOUT of the column to be removed from F_{11} . Similarly, **Find_Column_to_Add(IIN)** selects from among the indices of region (iii) of (5.4) the intrinsic index IIN of the column to be added to F_{11} . Finally, **Update_Factored_Kernel** updates the data structures used to represent F_{11} .

E. COMPLETE ALGORITHM DESCRIPTION

We are now in a position to fully describe our implementation of the algorithm. We do so by expanding the discussion of each step given previously in section C.

1. Initialize the algorithm using the origin as the initial basic solution. Then

$$\begin{aligned} B^0 &= (-I) \\ D^0 &= \begin{pmatrix} F \\ E \end{pmatrix} \\ D^0 P^0 &= \begin{pmatrix} F \\ E \end{pmatrix} \end{aligned}$$

2. Stop if the current solution is optimal. The current solution is optimal if $(RIM(IR) \geq 0 \text{ for } 1 \leq IR \leq M \text{ and } RIM(JC) \leq 0 \text{ for } M+1 \leq JC \leq M+N.$
3. (a) Select a violated primal or dual constraint as the target row or column, respectively. For purposes of exposition, assume the current solution is

primal feasible and we are executing the primal algorithm. Then the target row is the bottom row, and we select the intrinsic index IPCI of a column which will allow us to make a gain in the value of the extremal function (i.e., $RIM(IPCI) > 0$).

(b) **Generate_Column(IPCI).**

(c) Calculate ratios by computing $RIM(IR)/ROWCOL(IR)$ for those $1 \leq IR \leq M$ with $ROWCOL(IR) > 0$. If $ROWCOL(IR) \leq 0$ for all $IR, 1 \leq IR \leq M$ stop, since the problem is primal unbounded. Otherwise, select IPRI to be the intrinsic index yielding the smallest such ratio. Break ties in accordance with the following hierarchy: region (iii) first, then region (ii), then (i) and finally region (iv). Within a region, break ties by selecting IPRI to be the first such index encountered.

(d) **Generate_Row(IPRI).**

4. If, contrary to our assumption in 2.a. the current solution is not primal feasible, the target row would be some row IPRI of (5.4), $1 \leq IPRI \leq M$, rather than the bottom row. We would proceed by next executing **Generate_Row(IPRI)**. If $ROWCOL(JC) \geq 0$ for all $M+1 \leq JC \leq M+N$, we would conclude that the problem is primal infeasible.

5. (a) **Primary_Index_Exchange(IPRI, IPCI).**

(b) Pivot update $RIM()$ and OPT using $ROWCOL()$.

(c) Perform the necessary secondary and tertiary exchanges, as shown in Table 2. Note that some tertiary update actions are conditional, depending upon the singularity of F_{11} . We use a notation similar to the FORTRAN BLOCK IF statement to indicate the conditional actions.

(d) **Update_Factored_Kernel.**

(e) **Update_Explicit_Transformation_Kernel.**

(f) Go to Step 2.

We now give a more detailed explanation of the secondary and tertiary exchanges listed in Table 5.2, discussing each pivot coordinate individually.

1. Pivotal coordinate $((i), (j))$. EPCI, the extrinsic index of an E -type constraint located in position IPCI of B , is exchanged with EPRI, the extrinsic index of a nonnegativity constraint located in position IPRI of D . The initial pivot exchange places EPCI in position IPRI of (i) and EPRI in position IPCI of (j) , and thus secondary row and column exchanges are necessary to restore the structure of the tableau. Beginning with the column structure, we exchange column EPRI in position IPCI of region (j) with the extrinsic index located in position NXR of region (jjj) , and then decrement the value of NXR. This has the effect of shifting the starting column index of region (jj) one position to the left. Now extrinsic index EPRI is in position $(NXR+1)$ of the tableau and thus is still misplaced. We then exchange EPRI in position $(NXR+1)$ with the extrinsic index located in position NFR. This places EPRI contiguously with region (jjj) , so by decrementing NFR we restore the column factorization. The effect has been to decrease the dimension of (j) by one, increase the dimension of (jjj) by one and to shift the entire (jj) region one position to the left. In the row structure, EPCI in position IPRI of region (i) must eventually be moved to region (iv) . However, prior to this pivot EPRI was a key variable, located in region (i) . Its removal has disturbed the structure and thus the nonsingularity of F_{11} . We must therefore identify a column currently in region (iii) which can be used to restore the nonsingularity of F_{11} . We

thus invoke **Find_Column_to_Add(IIN)**, which identifies the intrinsic index IIN of such a column. Suppose the extrinsic index of the column in position IIN is KX. We exchange EPCI in position IPRI of region (i) with KX in position IIN of region (iii). We now have extrinsic index EPCI in position IIN of (iii) and we must move it to region (iv). We thus exchange EPCI in position IIN in region (iii) with the extrinsic index in position NXR of (iii), and then decrement NXR. This completes the restructuring of the row factorization with the effect of decreasing the dimension of region (iii) while increasing that of region (iv). The effect of these exchanges has been to reduce the dimension of \tilde{A}_{11}^{-1} through the deletion of column EPCI and row KX. To maintain the proper structure of \hat{A}_{11}^{-1} , we therefore **Delete_Row(KX)** and **Delete_Column(EPCI)**.

2. Pivotal coordinate ((i), (jj)). EPCI, the extrinsic index of a basic F -type constraint located in position IPCI of B , is exchanged with EPRI, the extrinsic index of a nonbasic nonnegativity constraint located in position IPRI of D . Since both EPCI and EPRI are misplaced after the primary exchange, secondary row and column exchanges are needed. In the columns, we exchange EPRI in position IPCI of region (jj) with the extrinsic index located in position NFR of region (jj). Decrementing NFR then completes the column exchanges, leaving the dimension of region (jj) reduced by one and that of (jjj) increased by one. In the rows, we exchange EPCI in position IPRI of region (i) with the extrinsic index in position MKC of region (i). Decrementing MKC then restores the row factored structure of the tableau. It remains to be seen, however, whether what remains of F_{11} after the removal of row EPCI and column EPRI is still nonsingular. We test F_{11} by invoking **Is_Factored_Kernel_Singular**.

If the response is FALSE, then no tertiary exchanges are required. If, however, the response is TRUE, we must identify a column exchange that we will restore the nonsingularity of F_{11} . To identify the two columns which will be involved in the exchange, we invoke **Find_Column_to_Remove**(IOUT) which returns the intrinsic index IOUT (with associated extrinsic index NX, say) of a column to be removed from F_{11} . We then call **Find_Column_to_Add**(IIN) which identifies the intrinsic index IIN (with associated extrinsic index KX, say) of a column in region (iii) which can be used to replace IOUT. We now invoke **Row_Index_Exchange**(IIN,IOUT), which completes the update of the row factorization. This row exchange results in the replacement of row KX of by row NX of the tableau. We currently have no representation of row KX of the tableau, so we compute it by invoking **Generate_Row**(IOUT). We may then restore the structure of \tilde{A}_{11}^{-1} by invoking **Replace_Row**(KX,NX).

3. Pivotal coordinate ((i),(jjj)). EPCI, the extrinsic index of a basic nonnegativity constraint located in position IPCI of B , is exchanged with extrinsic index EPRI located in position IPRI of D . The primary exchange properly places EPRI in region (jjj), and thus no secondary column exchanges are needed. Likewise, EPCI is properly placed in position IPRI of region (i), so no secondary row exchanges are required. Prior to the pivot, EPRI was designated a key column and its removal disturbed the structure of F_{11} . Since EPCI is itself a column, it is possible that it may be a replacement for EPRI. To determine if this is the case, we invoke **Is_Factored_Kernel_Singular**. If the response is FALSE, the direct exchange of column EPCI for column EPRI is justified and the tableau update is complete. If the response is TRUE, however, a tertiary row exchange is required. EPCI, currently located in position

IPRI of region (i) , will be the column we remove from F_{11} , and its replacement is found by invoking **Find_Column_to_Add(IIN)** which returns the intrinsic index IIN (associated with extrinsic index KX, say) located in region (iii) . We perform a tertiary exchange by invoking **Row_Index_Exchange(IIN,IPRI)**. This completes the row factorization update. This tertiary exchange results in the replacement of row KX of \hat{A}_{11}^{-1} by row EPCI of the tableau. We must therefore perform the corresponding row exchange in \hat{A}_{11}^{-1} also. To perform such an exchange, we require the current representation of row IPRI of the tableau. This is precisely the pivot row and is already available in the work array ROWCOL(). Thus, we may invoke **Replace_Row(KX,EPCI)** directly.

4. Pivotal coordinate $((ii),(j))$. EPCI, the extrinsic index of a basic E -type constraint located in position IPCI of B , is exchanged with EPRI, the extrinsic index of a nonbasic F -type constraint located in position IPRI of D . After the primary exchange, both EPCI (in (ii)) and EPRI (in (j)) are misplaced, and thus both secondary row and column exchanges are necessary. We exchange EPRI in position IPCI of region (j) with the extrinsic index in position NXR of region (j) and then decrement NXR. This has the effect of reducing the dimension of region (j) while increasing that of region (jj) . Notice that we have added row EPRI to the structure of F_{11} , and thus an additional column must at some point be identified. Several row exchanges are needed. First, we exchange row EPCI in position IPRI of region (ii) with the extrinsic index located in position (MKC+1) of region (ii) and then decrement MKC. This restores the structure of region (ii) by reducing its dimension by one row. We now must execute a series of exchanges that ultimately places EPCI in region (iv) and also adds a column

to region (i) to restore the nonsingularity of F_{11} . To do this, we first identify the column to be added to F_{11} by invoking **Find_Column_to_Add(IIN)**, which identifies the intrinsic index IIN in region (iii) (associated with an extrinsic index KX, say) which may be added to F_{11} . Notice that the dimension of \tilde{F}_{11} is increased by one through the addition of row EPRI and column KX. We then perform **Row_Index_Exchange(IIN,MKC)**, which places KX in position MKC of region (i) (correctly) and places EPCI in position IIN of region (iii) (incorrectly). To complete the tertiary row exchanges, we invoke **Row_Index_Exchange(IIN,MXR)**, followed by **Decrement(MXR)**. This properly places EPCI in region (iv) by increasing the dimension of region (iv) by one row while decreasing that of region (iii). Finally, notice that the dimension of \tilde{A}_{11}^{-1} has been reduced by one through the removal of column EPCI and row KX. To update the representation of \tilde{A}_{11}^{-1} accordingly, we invoke **Delete_Row(KX)** followed by **Delete_Column(EPCI)**.

5. Pivotal coordinate $((ii), (jj))$. EPCI, the extrinsic index of a basic F -type constraint located in position IPCI of B , is exchanged with EPRI, the extrinsic index of a nonbasic F -type constraint located in position IPRI of D . The primary exchange places both EPRI and EPCI in their proper positions, so no secondary row or column exchanges are needed. However, the structure of has been altered through the addition of row EPRI and the deletion of row EPCI. To determine if the structure and nonsingularity of F_{11} has remained intact, we invoke **Is_Factored_Kernel_Singular**. If the response is FALSE, the tableau update is complete. If the response is TRUE, we must identify a column exchange that will restore nonsingularity. Thus, we invoke **Find_Column_to_Remove(IOUT)** which returns the intrinsic index IOUT

(associated with the extrinsic index NX , say) of a column located in region (i) which may be removed from F_{11} . We invoke **Find_Column_to_Add**(IIN) which returns the intrinsic index (associated with an extrinsic index KX , say) of the column IIN located in region (iii) which may be added to F_{11} . We then perform a tertiary exchange by invoking **Row_Index_Exchange**(IIN, IOU). This last action results in the replacement of row IIN of \tilde{A}_{11}^{-1} by row IOU of the tableau. We have no current representation of row IOU, and to compute one we invoke **Generate_Row**(IOU). We then update the representation of \tilde{A}_{11}^{-1} by invoking **Replace_Row**(KX, NX).

6. Pivotal coordinate $((ii), (jjj))$. EPCI, the extrinsic index of a basic nonnegativity constraint located in position IPCI of B , is exchanged with EPRI, the extrinsic index of a nonbasic F -type constraint located in position IPRI of D . The primary exchange misplaces both EPRI and EPCI and thus both secondary row and column exchanges are necessary. We exchange EPRI, located in position IPCI of region (jjj) with the extrinsic index located in position NFR. By then incrementing NFR, we restore the column factorization structure by increasing the dimension of region (jj) while reducing that of region (jjj) . Notice that the addition of row EPRI to the structure of F_{11} indicates that an a corresponding column must also be located. To restore the structure of region (ii) , we exchange EPCI in position IPRI of region (ii) with the extrinsic index in position MXC of region (i) . Incrementing MKC restores the structure of region (ii) by reducing its dimension by one row. Since EPCI is a column which we have now placed in position MKC, it is possible that F_{11} is now nonsingular. To determine this, we invoke **Is_Factored_Kernel_Singular**. If the response is FALSE, the tableau is complete. If, on the other hand,

the response is TRUE, we must perform a tertiary column exchange to restore the nonsingularity of F_{11} . We will be removing EPCI, currently located in position MKC, from F_{11} and replacing it with the column identified by **Find_Column_to_Add(IIN)**, which is extrinsic index KX located in position IIN of region (iii). We then perform **Row_Index_Exchange(IIN,MKC)**, which completes the tertiary exchange. This exchange implies a row exchange in \hat{A}_{11}^{-1} also. We are required to replace row KX of \hat{A}_{11}^{-1} with the current representation of row EPCI of the tableau. Since IPRI is the pivot row, the current representation of EPCI is readily available in ROWCOL(). We may then complete the row exchange by invoking **Replace_Row(KX,EPCI)**.

7. Pivotal coordinate ((iii),(j)). EPCI, the extrinsic index of a basic E-type constraint located in position IPCI of B , is exchanged with EPRI, the extrinsic index of a nonbasic nonnegativity constraint located in position IPRI of D . The primary exchange misplaces both EPRI and EPCI, and thus both secondary row and column exchanges are necessary. We exchange EPRI, in position IPCI of region (j), with the extrinsic index in position NXR of region (j). We then exchange EPRI, now in position NXR of region (j), with the extrinsic index located in position NFR of region (jj). We complete the column update by decrementing both NXR and NFR, which has the effect of reducing the dimension of region (j) by one column, shifting region (jj) one column to the left in the tableau and increasing the dimension of region (jjj) by one column. For the row update, we first exchange EPCI, in position IPRI of region (iii), with the extrinsic index located in position MXR of region (iii). By then decrementing MXR we restore the row factorization by increasing the dimension of region (iv) by one row while decreasing that of region

(iii). The effect on \tilde{A}_{11}^{-1} has been to reduce its dimension by one through the deletion of column EPCI and row EPRI. The corresponding update to the representation of \tilde{A}_{11}^{-1} requires that we invoke **Delete_Row**(EPRI) followed by **Delete_Column**(EPCI).

8. Pivotal coordinate ((iii), (jj)). EPCI, the extrinsic index of a basic F-type constraint located in position IPCI of B , is exchanged with EPRI, the extrinsic index of a nonbasic nonnegativity constraint located in position IPRI of D . The primary exchange misplaces both EPRI and EPCI, and thus both secondary row and column exchanges are necessary. We exchange EPRI, currently located in position IPCI of region (jj), with the extrinsic index currently located in position NFR of region (jj). By then decrementing NFR, we restore the column factorization by decreasing the dimension of region (jj) while increasing that of region (jjj). Note that the removal of EPCI from the structure of F_{11} implies that the dimension of F_{11} decreases by one, and thus a corresponding column must be removed from F_{11} . To locate this column, we invoke **Find_Column_to_Remove**(IOUT), which returns the intrinsic index IOUT (associated with the extrinsic index NX, say) of a column located in region (i) which may be removed from F_{11} while allowing the remaining structure to be nonsingular. We exchange EPCI, currently located in position IPRI of region (iii), with NX, currently located in position IOUT of region (i). This restores the row structure of region (iii), but EPCI is misplaced in region (i). Therefore, we exchange EPCI in position IOUT of region (i) with the extrinsic index located in position MKC of region (i). By then decrementing MKC, we restore the structure of region (i) by decreasing its dimension and that of region (iii) by increasing its dimension. The exchange of NX in IIN with

EPCI in IPRI implies a row exchange in \tilde{A}_{11}^{-1} . To perform this exchange in the representation of \tilde{A}_{11}^{-1} , we must replace row EPRI of \tilde{A}_{11}^{-1} with the current representation of row NX of the tableau. Since this current representation is not available, we invoke **Generate_Row(IIN)** to compute it. We then update our representation of \tilde{A}_{11}^{-1} by invoking **Replace_Row(EPRI,NX)**.

9. Pivotal coordinate $((iii), (jj))$. EPCI, the extrinsic index of a basic nonnegativity constraint currently located in position IPCI of B , is exchanged with EPRI, the extrinsic index of a nonbasic nonnegativity constraint currently located in position IPRI of D . The primary exchange properly places EPRI and EPCI, so no secondary exchanges are necessary. Since F_{11} is unaffected, no additional action is required.
10. Pivotal coordinate $((iv), (j))$. EPCI, the extrinsic index of a basic E -type constraint currently located in position IPCI of B , is exchanged with EPRI, the extrinsic index of a nonbasic E -type constraint currently located in position IPRI of D . The primary exchange properly places EPRI and EPCI, so no secondary exchanges are necessary. F_{11} is unaffected, so no additional action is required.
11. Pivotal coordinate $((iv), (jj))$. EPCI, the extrinsic index of a basic F -type constraint currently located in position IPCI of B , is exchanged with EPRI, the extrinsic index of a nonbasic E -type constraint currently located in position IPRI of D . The primary exchange misplaces both EPRI and EPCI, so both secondary row and column exchanges are necessary. EPRI, currently in position IPCI of region (jj) , is exchanged with the extrinsic index located at position $(NXR+1)$ of region (jj) . The column structure is restored by incrementing NXR , increasing the dimension of region (j) while

decreasing that of region (jj) . Note that since the dimension of region (j) increases, the dimension of will increase also. Also, the removal of EPCI from F_{11} implies that the dimension of F_{11} will decrease. Thus, a column of F_{11} must be identified for removal as well. To find such a column, we invoke **Find_Column_to_Remove**(IOUT), which returns the intrinsic index IOUT (associated with the extrinsic index NX, say) of a column whose removal from F_{11} allows the remaining structure of F_{11} to be nonsingular. We perform a tertiary exchange by exchanging NX, currently located in position IOUT of region (i) , with the extrinsic index currently located in position MKC of region (i) . The structure of region (i) is then restored by decrementing MKC, reducing the dimension of region (i) while increasing that of region (ii) . To properly position EPCI and restore the structure of region (ii) , we exchange NX, currently located in position (MKC+1) of region (ii) , with EPCI, currently located in position IPRI of region (iv) . This leaves only NX misplaced. We thus exchange NX, currently located in position IPRI of region (iv) , with the extrinsic index located in position MXR of region (ii) . The row update is completed by incrementing MXR, increasing the dimension of region (iii) while decreasing that of region (iv) . Since the dimension of \hat{A}_{11}^{-1} has been increased, we must update our representation. We are required to add the current representation of column EPRI, which is available as the pivot column in ROWCOL(). We also require the current representation of row NX of the tableau, which is not currently available. We thus invoke **Generate_Row**(IOUT) With these two representations, we may then invoke **Add_Row**(NX) and **Add_Column**(EPCI) to complete the update of the representation of \hat{A}_{11}^{-1} .

12. Pivotal coordinate $((iv), (jjj))$. EPCI, the extrinsic index of a basic nonnegativity constraint currently located in position IPCI of B , is exchanged with EPRI, the extrinsic index of a nonbasic E -type constraint currently located in position IPRI of D . The primary exchange misplaces both EPRI and EPCI. To restore the column factorization, we first exchange EPRI, located in position IPCI of region (jjj) , with the extrinsic index located in position $(NFR+1)$ of region (jjj) . We then exchange EPCI, now in position $(NFR+1)$ of region (jjj) , with the extrinsic index located in position $(NXR+1)$. We complete the column update by incrementing both NXR and NFR , which has the effect of increasing the dimension of region (j) , shifting region (jj) one position to the right and decreasing the dimension of region (jjj) . Since the dimension of region (j) has been increased, the dimension of \tilde{A}_{11}^{-1} will increase as well. To restore the row factorization, we exchange EPCI, currently in position IPRI of region (iv) , with the extrinsic index located in position $(MXR+1)$ of region (iv) . The row update is completed by incrementing MXR , which has the effect of increasing the dimension of region (iii) while decreasing that of region (iv) . The dimension of \tilde{A}_{11}^{-1} has been increased through the addition of column EPCI and row EPRI. Since IPCI is the pivot column and IPRI is the pivot row, each is already available in ROWCOL(). Thus we simply invoke **Add_Row**(EPRI) followed by **Add_Column**(EPCI) to complete our update of the representation of A_{11}^{-1} .

TABLE 5.2: Secondary and Tertiary Tableau Exchanges

(i)	(v)	(vi)	(vii)
(i)	<pre> Column_Index.Exchange(LCP, NXR) Decrement(NXR) Column_Index.Exchange(NXR + 1, NFR) Find Column to Add(NIN) Row_Index.Exchange(LRP, NIN) Row_Index.Exchange(NIN, MVR) Decrement(MVR) Delete Row(KX) Delete Column(LC) </pre>	<pre> Column_Index.Exchange(LCP, NFR) Decrement(NFR) Row_Index.Exchange(LRP, MKC) If(IsFactoredKernel.Singular())then Find Column to Remove(ROUT) Row_Index.Exchange(NIN, ROUT) Generate a Row(ROUT) Replace Row(KX, NX) Endif </pre>	<pre> If(IsFactoredKernel.Singular())then Find Column to Add(NIN) Row_Index.Exchange(LRP, NIN) Replace Row(KX, LC) Endif </pre>
(ii)	<pre> Column_Index.Exchange(LCP, NXR) Decrement(NXR) Row_Index.Exchange(LRP, MKC + 1) Increment(MKC) Find Column to Add(NIN) Row_Index.Exchange(NIN, MKC) Row_Index.Exchange(NIN, MVR) Decrement(MVR) Delete Row(KX) Delete Column(LC) </pre>	<pre> If(IsFactoredKernel.Singular())then Find Column to Remove(ROUT) Generate a Row(ROUT) Find Column to Add(NIN) Row_Index.Exchange(NIN, ROUT) Replace Row(KX, NX) Endif </pre>	<pre> Column_Index.Exchange(NFR, LCP) Increment(NFR) Row_Index.Exchange(LRP, MKC) Increment(MKC) If(IsFactoredKernel.Singular())then Find Column to Add(NIN) Row_Index.Exchange(MKC, NIN) Replace Row(KX, LC) Endif </pre>
(iii)	<pre> Column_Index.Exchange(LCP, NXR) Column_Index.Exchange(NXR, NFR) Decrement(NXR) Decrement(NFR) Row_Index.Exchange(LRP, MVR) Decrement(MVR) Delete Row(LR) Delete Column(LC) </pre>	<pre> Column_Index.Exchange(LCP, NFR) Decrement(NFR) Find Column to Remove(ROUT) Generate a Row(ROUT) Row_Index.Exchange(ROUT, LRP) Row_Index.Exchange(ROUT, MKC) Decrement(MKC) Replace Row(LR, NX) </pre>	No Action Required
(iv)	No Action Required	<pre> Column_Index.Exchange(LCP, NXR + 1) Increment(NXR) Find Column to Remove(ROUT) Row_Index.Exchange(ROUT, MKC) Decrement(MKC) Row_Index.Exchange(MKC + 1, LRP) Row_Index.Exchange(LRP, MVR) Increment(MVR) Generate a Row(ROUT) Add Row(NX) Add Column(LC) </pre>	<pre> Column_Index.Exchange(NFR + 1, LCP) Increment(NFR) Column_Index.Exchange(NFR, NXR + 1) Increment(NXR) Row_Index.Exchange(LRP, MVR + 1) Increment(MVR) Add Row(LR) Add Column(LC) </pre>

VI. FACTORIZATION OF GENERALIZED UPPER BOUND ROWS

A. INTRODUCTION

We now develop the first of three specializations of the factorization approach. We are still interested in the problem:

$$\begin{aligned}
 \text{(GUB)} \quad \min_y \quad & wy \\
 \text{s.t. :} \quad & Fy \leq b \quad F \quad p \text{ by } n \\
 & Ey \leq r \quad E \quad m \text{ by } n \\
 & -Iy \leq 0 \quad -I \quad n \text{ by } n
 \end{aligned}$$

where $F, E, -I, w, b$ and r are as before. We now require that the F -type constraints are generalized upper bound (GUB) constraints. Define $S_i, i = 1, \dots, p$ to be pairwise disjoint subsets of the set $N = \{1, \dots, n\}$ and further define $S_0 = N - \bigcup_{i=1}^p S_i$ (S_0 may be empty). Then $S_i \cap S_{i'} = \emptyset$ for $0 \leq i \leq p, 0 \leq i' \leq p, i \neq i'$ and $\bigcup_{i=0}^p S_i = N$. GUB constraints are of the form

$$\sum_{j \in S_i} \delta_{ij} y_{ij} \leq b_i, \quad i = 1, \dots, p, \quad \delta_{ij} = \pm 1 \quad (6.1)$$

The sets $S_i, i = 0, \dots, p$ are called GUB sets and i is the GUB set index. S_0 identifies those variables that have no coefficient in any GUB constraint. The E -type constraints are of arbitrary form.

A specialization of the Simplex Method to handle GUB constraints was first developed by Dantzig and Van Slyke [1967]. They introduce a specialization of (GUB) with $\delta_{ij} = 1$ and strict equality constraints. Their algorithm requires a working basis of dimension $(m + 1)$. McBride [1972] develops a specialization of the mutual primal-dual method to solve a second variant of (GUB) with $\delta_{ij} = \pm 1$

and strict equality constraints. A dynamic working basis whose dimension cannot exceed m is required. A computational analysis by McBride [1972] predicts that the performance of this algorithm should exceed that of the Simplex Method when the proportion of GUB constraints in the model exceeds approximately 29%, and his empirical results tend to confirm this analysis.

We extend this work by developing a specialization of the factorization approach which does not require all GUB constraints to be binding. We will see that there are computational efficiencies to be gained by this approach. We first present the factored tableau for (GUB). We then discuss the important computational issues.

B. THE FACTORED TABLEAU

Recall from Chapter 4 the primal row basis at an arbitrary point in the solution process is:

$$B = \begin{array}{c} (j) \\ (jj) \\ (jjj) \end{array} \left(\begin{array}{ccc} \overbrace{E_{11} \dots E_{1k}}^k & \overbrace{E_{1k+1} \dots E_{1l}}^l & \overbrace{E_{1l+1} \dots E_{1n}}^{n-l-k} \\ \overbrace{F_{11} \dots F_{1k}}^k & \overbrace{F_{1k+1} \dots F_{1l}}^l & \overbrace{F_{1l+1} \dots F_{1n}}^{n-l-k} \\ 0 & 0 & -I \end{array} \right) \begin{array}{l} \} l \\ \} k \\ \} n-l-k \end{array} \quad (6.2)$$

We saw in Chapter 4 that it is always possible to identify among the columns of B a set of k columns such that the submatrix F_{11} is nonsingular. Since the rows of $[F_{11} \ F_{12} \ F_{13}]$ correspond to GUB constraints, their form is similar to (6.1), with column permutations accounting for the difference. Each column of $[F_{11} \ F_{12} \ F_{13}]$ is either zero, or a signed unit vector, and thus by row permutations of these F -type constraints, F_{11} may be placed into the form of a k by k signed identity matrix Δ_{11} . The resulting (GUB) row basis is:

$$B = \begin{matrix} & \overbrace{\quad k \quad} & \overbrace{\quad l \quad} & \overbrace{\quad n-l-k \quad} \\ (j) & \begin{pmatrix} E_{11} & E_{12} & E_{13} \end{pmatrix} & \} l \\ (jj) & \begin{pmatrix} \Delta_{11} & F_{12} & F_{13} \end{pmatrix} & \} k \\ (jjj) & \begin{pmatrix} 0 & 0 & -I \end{pmatrix} & \} n-l-k \end{matrix} \quad (6.3)$$

The corresponding nonbasic constraint rows are then:

$$D = \begin{matrix} & \overbrace{\quad k \quad} & \overbrace{\quad l \quad} & \overbrace{\quad n-l-k \quad} \\ (i) & \begin{pmatrix} -I & 0 & 0 \end{pmatrix} & \} k \\ (ii) & \begin{pmatrix} 0 & F_{22} & F_{23} \end{pmatrix} & \} p-k \\ (iii) & \begin{pmatrix} 0 & -I & 0 \end{pmatrix} & \} l \\ (iv) & \begin{pmatrix} E_{21} & E_{22} & E_{23} \end{pmatrix} & \} m-l \end{matrix} \quad (6.4)$$

Recall that we wish to observe a one-to-one association between the variables whose nonnegativity constraints (which appear in region (i) of D) are nonbasic (nonbinding) and the F -type constraints in region (jj) of B . This association is exactly analogous to the idea of "key" variables proposed by Dantzig and Van Slyke [1967]. We call the variables whose nonnegativity constraints appear in region (i) of D "key" variables, and there is one such variable for each currently basic F -type constraint appearing in region (jj) of B . The variables whose nonnegativity constraints appear in region (iii) of D are then called "non-key".

Using (6.3) and (6.4) and recalling the expression for the explicit transformation kernel:

$$\hat{A}_{11}^{-1} \hat{\triangleq} (E_{12} - E_{11}F_{11}^{-1}F_{12})^{-1} = (E_{12} - E_{11}\Delta_{11}F_{12})^{-1}$$

the principal part of the factored tableau is:

$$\begin{array}{l}
 \begin{array}{c} \text{(i)} \\ \text{(ii)} \\ \text{(iii)} \\ \text{(iv)} \end{array} \left(\begin{array}{ccc} \underbrace{\quad}_{(i)} & \underbrace{\quad}_{(ii)} & \underbrace{\quad}_{(iii)} \\ -\Delta_{11}F_{12}\hat{A}_{11}^{-1} & \Delta_{11} + \Delta_{11}F_{12}\hat{A}_{11}^{-1}E_{11}\Delta_{11} & \Delta_{11}F_{12} - \Delta_{11}F_{12}\hat{A}_{11}^{-1}(E_{12} - E_{11}\Delta_{11}F_{12}) \\ -F_{22}\hat{A}_{11}^{-1} & F_{22}\hat{A}_{11}^{-1}E_{11}\Delta_{11} & F_{22} - F_{22}\hat{A}_{11}^{-1}(E_{12} - E_{11}\Delta_{11}F_{12}) \\ \hat{A}_{11}^{-1} & -\hat{A}_{11}^{-1}E_{11}\Delta_{11} & \hat{A}_{11}^{-1}(E_{12} - E_{11}\Delta_{11}F_{12}) \\ -(E_{22} - E_{22}\Delta_{11}F_{12})\hat{A}_{11}^{-1} & -E_{22}\Delta_{11} + (E_{22} - E_{22}\Delta_{11}F_{12})\hat{A}_{11}^{-1}E_{11}\Delta_{11} & E_{22} - E_{22}\Delta_{11}F_{12} - E_{22}\hat{A}_{11}^{-1}(E_{12} - E_{11}\Delta_{11}F_{12}) \\ & & + E_{22}\Delta_{11}F_{12}\hat{A}_{11}^{-1}(E_{12} - E_{11}\Delta_{11}F_{12}) \end{array} \right)
 \end{array}
 \tag{6.5}$$

C. SEQUENCING COMPUTATIONS TO EXPLOIT COMMON SUB-EXPRESSIONS IN COLUMN GENERATION

To examine the actions required by **Generate_Column(LC)**, suppose we want to generate column LC of the principal part of the tableau (6.5) and place the results in the vector $z^T = (z_1, z_2, z_3, z_4)^T$ which is partitioned to conform to (6.5). Rewriting (6.5) in a more convenient manner, we have:

$$\begin{array}{l}
\text{(i)} \quad \underbrace{\quad}_{(i)} \quad \underbrace{\quad}_{(ii)} \quad \underbrace{\quad}_{(iii)} \\
\text{(ii)} \quad \left(\begin{array}{ccc}
-\Delta_{11} F_{12} [\hat{A}_{11}^{-1}] & -\Delta_{11} F_{12} [-\hat{A}_{11}^{-1} E_{11} \Delta_{11}] + \Delta_{11} & -\Delta_{11} F_{12} [\hat{A}_{11}^{-1} (E_{13} - E_{11} \Delta_{11} F_{13})] + \Delta_{11} F_{13} \\
-F_{22} [\hat{A}_{11}^{-1}] & -F_{22} [-\hat{A}_{11}^{-1} E_{11} \Delta_{11}] & -F_{22} [\hat{A}_{11}^{-1} (E_{13} - E_{11} \Delta_{11} F_{13})] + F_{23} \\
\hat{A}_{11}^{-1} & -\hat{A}_{11}^{-1} E_{11} \Delta_{11} & \hat{A}_{11}^{-1} (E_{13} - E_{11} \Delta_{11} F_{13}) \\
-E_{22} [\hat{A}_{11}^{-1}] - E_{21} [-\Delta_{11} F_{12} (\hat{A}_{11}^{-1})] & -E_{22} [-\hat{A}_{11}^{-1} E_{11} \Delta_{11}] & -E_{22} [\hat{A}_{11}^{-1} (E_{13} - E_{11} \Delta_{11} F_{13})] \\
-E_{21} [-\Delta_{11} F_{12} (-\hat{A}_{11}^{-1} E_{11} \Delta_{11}) + \Delta_{11}] & -E_{21} [-\Delta_{11} F_{12} (\hat{A}_{11}^{-1} (E_{13} - E_{11} \Delta_{11} F_{13})) + \Delta_{11} F_{13}] + E_{23}
\end{array} \right)
\end{array}
\quad (6.6)$$

To highlight those computations involving terms consisting entirely of zeros and plus and minus ones, we introduce the following notation: a general matrix product is denoted by “ \cdot ”, as in $\hat{A}_{11}^{-1} \cdot E_{13}$. A “simple” matrix product (i.e., one in which one of the terms consists entirely of zeros, plus ones and minus ones) is denoted by \diamond , as in $E_{11} \diamond F_{12}$. The calculations may then proceed as follows (note that ϵ^{LC} is a unit vector with a plus one in position LC):

1. Calculate region (iii):

(a) If column LC is in (j), then:

$$z_3 = \hat{A}_{11}^{-1} \diamond \epsilon^{LC} = (\hat{A}_{11}^{-1})^{LC}$$

(b) If column LC is in (jj), then:

$$z_3 = -\hat{A}_{11}^{-1} \cdot (E_{11}) \diamond (\Delta_{11})^{LC}$$

(c) If column LC is in (jjj) , then:

$$z_3 = \tilde{A}_{11}^{-1} \cdot [(E_{13})^{LC} - E_{11} \diamond \Delta_{11} \diamond (F_{13})^{LC}]$$

Notice that $(F_{13})^{LC}$ is either null or it is a signed unit vector, say in row k , in which case $E_{11} \diamond \Delta_{11} \diamond (F_{13})^{LC} = \pm(E_{11})^k$.

2. Calculate region (i):

(a) If column LC is in (j) , then:

$$z_1 = -\Delta_{11} \diamond F_{12} \diamond z_3$$

(b) If column LC is in (jj) , then:

$$z_1 = -\Delta_{11} \diamond F_{12} \diamond z_3 + (\Delta_{11})^{LC}$$

(c) If column LC is in (jjj) , then:

$$z_1 = -\Delta_{11} \diamond F_{12} \diamond z_3 + \Delta_{11} \diamond (F_{13})^{LC}.$$

Notice that the computation $-F_{12} \diamond z_3$ involves only additions and subtractions, and that each case in region (i) differs only by an addition suffix.

3. Calculate region (ii):

(a) If column LC is in (j) , then:

$$z_2 = -F_{22} \diamond z_3$$

(b) If column LC is in (jj) , then:

$$z_2 = -F_{22} \diamond z_3$$

(c) If column LC is in (jjj) , then:

$$z_2 = -F_{22} \diamond z_3 + (F_{23})^{LC}$$

4. Calculate region (iv) :

(a) If column LC is in (j) , then:

$$z_4 = -E_{21} \cdot z_1 - E_{22} \cdot z_3$$

(b) If column LC is in (jj) , then:

$$z_4 = -E_{21} \cdot z_1 - E_{22} \cdot z_3$$

(c) If column LC is in (jjj) , then:

$$z_4 = -E_{21} \cdot z_1 - E_{22} \cdot z_3 + (E_{23})^{LC}$$

Then column LC of (6.6) is available as $z^T = (z_1, z_2, z_3, z_4)^T$. Notice that floating point multiplications and/or divisions are necessary only for computation of regions (iii) and (iv); also note that regions (i), (ii) and (iv) are linear combinations of region (iii) and one another, and that each term in these regions differs only by an additive suffix.

D. SEQUENCING COMPUTATIONS TO EXPLOIT COMMON SUB-EXPRESSIONS IN ROW GENERATION

Now we examine the actions of **Generate_Row**(LR). Suppose we desire to generate row LR and place the results of the computation in the vector $\hat{z} = (\hat{z}_5, \hat{z}_6, \hat{z}_7)$ which is partitioned to conform to (6.5). Rewriting (6.5) in a convenient form:

$$\begin{array}{l}
 \text{(i)} \quad \left(\begin{array}{ccc} \underbrace{-\Delta_{11} F_{12} \hat{A}_{11}^{-1}} & \underbrace{-\{-\Delta_{11} F_{12} \hat{A}_{11}^{-1}\} E_{11} - I \} \Delta_{11}} & \underbrace{[-\Delta_{11} F_{12} \hat{A}_{11}^{-1}] E_{13} - \{-\Delta_{11} F_{12} \hat{A}_{11}^{-1}\} E_{11} - I \} \Delta_{11} \} F_{13}} \\
 \text{(ii)} \quad \left(\begin{array}{ccc} \underbrace{-F_{22} \hat{A}_{11}^{-1}} & \underbrace{-\{-F_{22} \hat{A}_{11}^{-1}\} E_{11} \} \Delta_{11}} & \underbrace{[-F_{22} \hat{A}_{11}^{-1}] E_{13} + \{-F_{22} \hat{A}_{11}^{-1}\} E_{11} \} \Delta_{11} \} F_{13} + F_{22}} \\
 \text{(iii)} \quad \left(\begin{array}{ccc} \underbrace{\hat{A}_{11}^{-1}} & \underbrace{-\{\hat{A}_{11}^{-1}\} E_{11} \} \Delta_{11}} & \underbrace{[\hat{A}_{11}^{-1}] E_{13} + \{-\{\hat{A}_{11}^{-1}\} E_{11} \} \Delta_{11} \} F_{13}} \\
 \text{(iv)} \quad \left(\begin{array}{ccc} \underbrace{(E_{11} \Delta_{11} F_{12} - E_{22} \hat{A}_{11}^{-1})} & \underbrace{-\{(E_{11} \Delta_{11} F_{12} - E_{22} \hat{A}_{11}^{-1}) E_{11} + E_{22}\} \Delta_{11}} & \underbrace{\{(E_{11} \Delta_{11} F_{12} - E_{22} \hat{A}_{11}^{-1}) E_{13} + \{(E_{11} \Delta_{11} F_{12} - E_{22} \hat{A}_{11}^{-1}) E_{11} + E_{22}\} \Delta_{11} \} F_{13} - E_{22}}
 \end{array} \right)
 \end{array} \right)
 \end{array}
 \end{array}
 \quad (6.7)$$

Then we may proceed as follows (note that e_{LR} is a unit vector with a plus one in position LR):

1. Calculate region (j):

(a) If row LR is in (i), then:

$$\tilde{z}_5 = [-(\Delta_{11})_{LR} \diamond F_{12}] \diamond \tilde{A}_{11}^{-1}$$

(b) If row LR is in (ii), then:

$$\tilde{z}_5 = (-F_{22})_{LR} \diamond \tilde{A}_{11}^{-1}$$

(c) If row LR is in (iii), then:

$$\tilde{z}_5 = e_{LR} \diamond \tilde{A}_{11}^{-1} = (\tilde{A}_{11}^{-1})_{LR}$$

(d) If row LR is in (iv), then:

$$\tilde{z}_5 = [(E_{21})_{LR} \diamond \Delta_{11} \diamond F_{12} - (E_{22})_{LR}] \cdot \tilde{A}_{11}^{-1}$$

Notice that the computation of the term $(E_{21})_{LR} \diamond \Delta_{11} \diamond F_{12} - (E_{22})_{LR}$ involves only additions and subtractions.

2. Calculate region (jj):

(a) If row LR is in (i), then:

$$\tilde{z}_6 = (-\tilde{z}_5 \cdot E_{11} + e_{LN}) \diamond \Delta_{11}$$

(b) If row LR is in (ii), then:

$$\tilde{z}_6 = (-\tilde{z}_5 \cdot E_{11}) \diamond \Delta_{11}$$

(c) If row LR is in (iii), then:

$$\tilde{z}_6 = (-\tilde{z}_5 \cdot E_{11}) \diamond \Delta_{11}$$

(d) If row LR is in (iv), then:

$$\tilde{z}_6 = (-\tilde{z}_5 \cdot E_{11} - (E_{21})_{LR}) \diamond \Delta_{11}$$

3. Calculate region (jjj):

(a) If row LR is in (i), then:

$$\tilde{z}_7 = \tilde{z}_5 \cdot E_{13} + \tilde{z}_6 \diamond F_{13}$$

(b) If row LR is in (ii), then:

$$\tilde{z}_7 = \tilde{z}_5 \cdot E_{13} + \tilde{z}_6 \diamond F_{13} + (F_{23})_{LR}$$

(c) If row LR is in (iii), then:

$$\tilde{z}_7 = \tilde{z}_5 \cdot E_{13} + \tilde{z}_6 \diamond F_{13}$$

(d) If row LR is in (iv), then:

$$\tilde{z}_7 = \tilde{z}_5 \cdot E_{13} + \tilde{z}_6 \diamond F_{13} + (E_{23})_{LR}.$$

The row LR of (6.7) is available as $\tilde{z} = (\tilde{z}_5, \tilde{z}_6, \tilde{z}_7)$. Note that floating point multiplications and/or divisions are avoided for terms involving F_{11} , F_{22} and F_{23} , that regions (jj) and (jjj) are linear combinations of region (j) , and that many cases differ only by additive infix terms.

This computational scheme extends the approach of the mutual primal-dual method by introducing additional linear dependencies into the tableau and exploits those dependencies to improve efficiency. Empirical evidence suggests that this approach also improves numerical stability (McBride [1989]).

E. DATA STRUCTURES

The essential information contained in the factored kernel when the factored rows are GUB constraints is simply the unique mapping between each basic factored constraint and its corresponding "key" nonbasic variable. We require a representation of this mapping which is compact, can be efficiently accessed and is easily updated.

One possibility is to maintain the intrinsic ordering of the tableau in such a way that the factored row/key variable relationship can be derived implicitly. Region (jj) of the tableau corresponds to the basic factored constraints of B . Region (i) of the tableau corresponds to the nonnegativity constraints associated with the key variables. If we maintain the ordering of (jj) and (i) so that the k^{th} position of (i) contains the index of the key variable associated with the factored constraint whose index is in the k^{th} position of (jj) , we have an implicit representation of F_{11} .

Two difficulties arise with such an approach which lead us to choose an alternate representation. First, such an ordering complicates the tableau update scheme presented in Chapter 5 and potentially increases the computational expense of all tableau updates which affect F_{11} . Second, several update cases require the identification of GUB set membership for variables which are not currently assigned to region (i) and thus are not currently part of the F_{11} structure. This may be done by accessing the original problem data by column and scanning that column for the index of the GUB row, if any, in which the column has a nonzero element. Since the original problem data is stored in a super-sparse form, this scheme requires indirect computer memory addressing. We prefer a method which requires less computational expense.

We thus introduce an additional data structure to manage the structure of F_{11} . $KEY(IJ)$ is an array of type INTEGER, dimensioned from 1 to $(m + n)$, which for each basic factored constraint records the extrinsic index of the associated key variable, and for each nonbasic key variable records the extrinsic index of the corresponding basic factored constraint. Additionally, for each variable which is not key (i.e., either non-key variables or basic variables), $KEY(IJ)$ records the extrinsic index of the factored constraint for which it may serve as a key variable. If we interpret the extrinsic index of the factored constraints as GUB set indices, then for every variable $KEY(IJ)$ contains the GUB set index. Note that since GUB set membership is fixed for a given variable, all column indices $KEY(IJ)$ can be initialized once and need never be updated. Only the factored constraint indices require updating.

F. FACTORED KERNEL UPDATE ACTIONS

Recall from Chapter 5 the details of the factored kernel update actions

Is_Factored_Kernel_Singular,
Find_Column_To_Remove(*IFKC*),
Find_Column_To_Add(*IFKC*), and
Update_Factored_Kernel,

are factorization-specific. We now discuss these details for the GUB factorization.

The row-permuted diagonal structure of F_{11} (Δ_{11}) implies that the question of singularity arising from rank-one updates can be resolved strictly on the basis of local information. That is, the exchange, deletion or addition of a row leads immediately to the identification of a unique GUB set. The search for a corresponding column (variable) to complete the exchange, deletion or addition is then limited to those sharing this GUB set membership. The simplicity of this structure greatly enhances the efficiency of the necessary update actions.

For example, consider pivot update case $((ii), (jj))$, in which basic factored constraint EPCI, located in tableau position IPCI is removed from the row basis and nonbasic factored constraint EPRI, located in tableau position IPRI replaces it. Because EPCI was basic, there is a key variable, say EPCIKV, located in position IPCIKV of region (i) of the tableau. Because GUB sets are, by definition, pairwise disjoint, EPCIKV may not serve as a key variable for the new basic factored constraint EPRI. Thus, a replacement variable among those in region (iii) of the tableau must be found. Such a replacement must exist, for otherwise B is singular. To locate such a variable, we scan the variables in region (iii), searching for one which is a member of GUB set EPRI. The test is simply this: for each variable JC in region (iii), if

$$\text{KEY}(\text{JC}) = \text{EPRI}.$$

then accept JC as the key variable for EPRI and proceed with the secondary tableau update exchange. Otherwise, continue the scan.

Thus, the action **Find_Column_To_Add(EKV)** scans the indices of region (iii), performing the GUB set membership test on each index. The first such index for which the test is true is chosen as EKV.

The action **Find_Column_To_Remove(EKV)** is required when a constraint ERI is removed from F_{11} . EKV is then the key variable corresponding to ERI and is determined by:

$$\text{EKV} = \text{KEY}(\text{ERI}).$$

Is_Factored_Kernel_Singular is required when the constraint EPRI is to be added to F_{11} and the pivot column index EPCI corresponds to a variable. Thus, EPCI is a possible key variable for EPRI and is a convenient candidate. It may be accepted as the key variable for EPRI if and only if F_{11} remains nonsingular after the addition of both EPRI and EPCI, and this is the case if EPCI is a member of GUB set EPRI. Thus, **Is_Factored_Kernel_Singular** simply compares $\text{KEY}(\text{EPCI})$ and EPRI. If they are unequal, F_{11} is singular and the result "true" is returned. Otherwise, the result is "false".

Finally, we consider **Update_Factored_Kernel**. Since the GUB set membership of a variable does not change, only constraint index updates of $\text{KEY}(\text{IR})$ are required. For example, if the old key variable index for constraint EPRI is EPRIKVO and the new index is EPRIKVN, the required update is:

$$\text{KEY}(\text{EPRI}) = \text{EPRIKVN}.$$

VII. FACTORIZATION OF PURE NETWORK ROWS

A. INTRODUCTION

We remain interested in the problem:

$$\begin{aligned}
 (\text{PNSC}) \quad \min_y : \quad & wy \\
 \text{s.t. :} \quad & Fy \leq b \quad ; \quad F \text{ } p \text{ by } n \\
 & Ey \leq r \quad ; \quad E \text{ } m \text{ by } m \\
 & -Iy \leq 0 \quad ; \quad -I \text{ } n \text{ by } m
 \end{aligned}$$

where F , E , $-I$, w , b and r are as before. We now require that each column of F consist of zero, one or two nonzero elements. If a column contains a single nonzero element within the rows of F , it may be either a plus one or a minus one. If a column contains two nonzero elements within the rows of F , one must be a plus one and the other a minus one. F -type constraints may also be interpreted as defining the node-arc incidence matrix of a directed graph. Define $\mathcal{I} = \{0, \dots, p\}$, $\mathcal{J} = \{1, \dots, n\}$, and $\mathcal{N} = \{n_i, i \in \mathcal{I} - \{0\}\}$ to be a set of nodes and $\mathcal{A} = \{a^k, k \in \mathcal{J} \mid a^k = (n_i, n_j), i \in \mathcal{I}, j \in \mathcal{I}\}$ to be a set of arcs with ordered pairs of nodes (tail, head) as elements indexed by k . Note that we interpret node 0 as a null node, and thus an arc incident to node 0 is viewed as a single-ended arc. Then a graph is defined as $\mathcal{G} = \{\mathcal{N}, \mathcal{A}\}$. The node-arc incidence matrix of \mathcal{G} is a matrix F consisting of p rows (one for each node in \mathcal{N}) and n columns (one for each arc in \mathcal{A}) with elements:

$$f_{i,k} = \begin{cases} +1 & \text{if } a^k = (n_i, n_j) \text{ for some } n_j \in \mathcal{N} \\ -1 & \text{if } a^k = (n_j, n_i) \text{ for some } n_j \in \mathcal{N} \\ 0 & \text{otherwise} \end{cases}$$

Thus, if arc $a^k = (n_i, n_j)$ is represented by column k of F , then

$$a^k = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ -1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \begin{matrix} i^{th} \text{ row} \\ \\ j \text{ row} \end{matrix},$$

and each column of F is either all zeros, contains exactly one nonzero element (which may be either plus one or minus one) or contains exactly two nonzero elements (a plus one and a minus one).

A specialization of (PNSC) that has been widely studied arises when the F -type constraints are taken to be equalities and the E -type constraints are vacuous, resulting in:

$$\begin{aligned} \text{(PNE)} \quad \min_y \quad & wy \\ \text{s.t.} \quad & Fy = b \quad F \text{ } p \text{ by } n \\ & -Iy \leq 0 \quad -I \text{ } n \text{ by } n \end{aligned}$$

Very efficient specializations of the primal Simplex Method have been developed and implemented to solve (PNE) (e.g., Srinivasan and Thompson [1973], Glover, et. al. [(1974)], Bradley, Brown and Graves [1977]). These algorithms exploit some well known properties of F (we assume that one redundant row has been removed from F):

1. Every primal Simplex basis B of F (consisting of $(p-1)$ linearly independent columns of F) can be triangulated by row and column permutations.

2. Every such basis B is itself the node-arc incidence matrix of a subgraph of \mathcal{G} , and this subgraph is a rooted spanning tree, and
3. F is totally unimodular, implying that if b_1 and b_2 are integer $(p-1)$ -vectors and x_1 and x_2 are $(p-1)$ -vectors of unknowns, then for every primal Simplex basis B of F the solutions of $Bx_1 = b_1$ and $x_2^T B = b_2^T$ are also integer.

Property (1) allows very efficient execution of Simplex iterations, property (2) motivates the use of special data structures which allow efficient storage and update of Simplex bases and property (3) allows all computations to be performed in integer arithmetic (assuming w and b are integer).

Several contributions have been made to solving variations of (PNSC) when the E -type constraints are not vacuous. The e approaches have their roots in work by Kaul [1965] and Bennett [1966]. Chen and Saigal considered a version of (PNSC) with all equality constraints. Hartman and Lasdon [1972] considered a specialization of (PNSC) to the multicommodity capacitated transshipment problem (MCTP), in which the E -type constraints are generalized upper bound (GUB) constraints and the F -type constraints decouple into independent pure network subproblems. In each of these treatments, the resulting algorithm is a specialization of the primal Simplex Method. McBride [1972] considered (PNSC) in the context of the mutual primal-dual method, with all constraints assumed to be equalities. Each of these approaches allows a basis representation which may be factored into two components: an explicit part of dimension m by m and a factored part of dimension p by p .

Our inequality formulation of (PNSC) allows a dynamic basis representation where, just as in the GUB specialization, both the dimension of the factored part and the dimension of the explicit part may vary from one iteration to the next.

B. THE FACTORED TABLEAU

Recall from Chapter 3 the primal row basis at an arbitrary point in the solution process is:

$$B = \begin{matrix} (j) \\ (jj) \\ (jjj) \end{matrix} \begin{pmatrix} E_{11} & E_{12} & E_{13} \\ F_{11} & F_{12} & F_{13} \\ 0 & 0 & -I \end{pmatrix} \quad (7.1)$$

The corresponding nonbasic constraint rows are then:

$$D = \begin{matrix} (i) \\ (ii) \\ (iii) \\ (iv) \end{matrix} \begin{pmatrix} -I & 0 & 0 \\ F_{21} & F_{22} & F_{23} \\ 0 & -I & 0 \\ E_{21} & E_{22} & E_{23} \end{pmatrix} \quad (7.2)$$

and the principal part of the factored tableau is:

$$\begin{matrix} (i) & (ii) & (iii) \\ \begin{pmatrix} (i) & (ii) & (iii) \\ (ii) & (iii) & (iv) \\ (iii) & (iv) & (v) \\ (iv) & (v) & (vi) \end{pmatrix} & \begin{pmatrix} (i) & (ii) & (iii) \\ (ii) & (iii) & (iv) \\ (iii) & (iv) & (v) \\ (iv) & (v) & (vi) \end{pmatrix} & \begin{pmatrix} (i) & (ii) & (iii) \\ (ii) & (iii) & (iv) \\ (iii) & (iv) & (v) \\ (iv) & (v) & (vi) \end{pmatrix} \end{matrix} \quad (7.3)$$

The F -type constraints represent the node-arc incidence matrix of a graph. For notational convenience, let us define the corresponding directed graph explicitly as $\mathcal{G} = \{\mathcal{N}, \mathcal{A}\}$ where \mathcal{G}, \mathcal{N} and \mathcal{A} are as previously defined. Note that while we choose to interpret all columns of (PNSC) as arcs in \mathcal{G} , any number of these columns may be

null with respect to the F -type constraints and thus may be interpreted graphically as null arcs.

Since the rows of $[F_{11} \ F_{12} \ F_{13}]$ in the primal row basis B form a subset of the rows of F , $[F_{11} \ F_{12} \ F_{13}]$ may itself be interpreted as the node-arc incidence matrix of a graph $\mathcal{G}_B = \{\mathcal{N}_B, \mathcal{A}_B\}$. \mathcal{N}_B is a subset of \mathcal{N} , but in general $\mathcal{A}_B \not\subset \mathcal{A}$. To see this, define $\mathcal{N}_D = \mathcal{N} - \mathcal{N}_B$ and consider an arc $a^q = (n_i, n_j)$ where $n_i \in \mathcal{N}_B$ and $n_j \in \mathcal{N}_D$. Then a^q "spans" the partitioning of \mathcal{N} into \mathcal{N}_B and \mathcal{N}_D . While a^q is a doubleton arc with respect to \mathcal{G} , it is a singleton arc with respect to \mathcal{G}_B . We call such an arc a "dynamic singleton", and we will discover that such arcs require special handling in our implementation.

Since a nonsingular F_{11} must exist, it is well known that the columns of F_{11} represent a rooted spanning tree defined over the nodes of \mathcal{N}_B , and that F_{11} may be placed into upper triangular form by row and column permutations. If we choose to represent F_{11} rather than F_{11}^{-1} in our implementation, we are interested in performing two fundamental operations:

1. Solving linear systems of the type

$$F_{11}z_1 = b_1$$

and

$$z_2^T F_{11} = b_2^T$$

where z_1 and z_2 are unknown and b_1 and b_2 are rationals (not necessarily integers), and

2. Placing \hat{F}_{11} in upper triangular form, where \hat{F}_{11} results from a column exchange, a row exchange, a column and row addition or a column and row deletion performed on F_{11} .

The literature on (PNE) demonstrates that with a proper choice of data structures, operation (1) may be performed very efficiently when b_1 and b_2 are integer, and that (2) may be done efficiently when the class of updates is limited to column exchanges. We will extend these existing approaches to deal with (1) and (2) in the (PNSC) setting.

C. SEQUENCING COMPUTATIONS TO EXPLOIT COMMON SUB-EXPRESSIONS IN COLUMN GENERATION

To examine the actions required by **Generate_Column(LC)**, suppose we want to generate column LC of the principal part of the tableau (7.3) and place the results in the vector $z^T = (z_1, z_2, z_3, z_4)^T$ which is partitioned to conform to (7.3). Rewriting (7.3) in a more convenient manner, we have:

$$\begin{array}{l}
 \text{i)} \quad \left[\begin{array}{ccc} \overbrace{-F_{11}^{-1} F_{12} [\hat{A}_{11}^{-1}]} & \overbrace{-F_{11}^{-1} F_{12} [-\hat{A}_{11}^{-1} E_{11} F_{11}^{-1}] + F_{11}^{-1}} & \overbrace{-F_{11}^{-1} F_{12} [\hat{A}_{11}^{-1} E_{13} - E_{11} F_{11}^{-1} F_{13}] + F_{11}^{-1} F_{13}} \\ \text{ii)} \quad \left[\begin{array}{ccc} \overbrace{-F_{21} [\hat{A}_{11}^{-1}] - F_{21} [-F_{11}^{-1} F_{12} \hat{A}_{11}^{-1}]} & \overbrace{-F_{21} [-\hat{A}_{11}^{-1} E_{11} F_{11}^{-1}]} & \overbrace{-F_{21} [\hat{A}_{11}^{-1} E_{13} - E_{11} F_{11}^{-1} F_{13}] + F_{21}} \\ \text{iii)} \quad \left[\begin{array}{ccc} \overbrace{-F_{21} [-F_{11}^{-1} F_{12} - \hat{A}_{11}^{-1} E_{11} F_{11}^{-1}] + F_{11}^{-1}} & \overbrace{-F_{21} [-F_{11}^{-1} F_{12} \hat{A}_{11}^{-1} E_{13} - E_{11} F_{11}^{-1} F_{13}] - F_{11}^{-1} F_{13}} \\ \text{iv)} \quad \left[\begin{array}{ccc} \overbrace{-E_{21} [\hat{A}_{11}^{-1}] - E_{21} [-F_{11}^{-1} F_{12} \hat{A}_{11}^{-1}]} & \overbrace{-E_{21} [-\hat{A}_{11}^{-1} E_{11} F_{11}^{-1}]} & \overbrace{-E_{21} [\hat{A}_{11}^{-1} E_{13} - E_{11} F_{11}^{-1} F_{13}] + E_{21}} \\ \text{v)} \quad \left[\begin{array}{ccc} \overbrace{-E_{21} [-F_{11}^{-1} F_{12} - \hat{A}_{11}^{-1} E_{11} F_{11}^{-1}] + F_{11}^{-1}} & \overbrace{-E_{21} [-F_{11}^{-1} F_{12} \hat{A}_{11}^{-1} E_{13} - E_{11} F_{11}^{-1} F_{13}] - F_{11}^{-1} F_{13}} \end{array} \right. \end{array} \right. \end{array} \right.
 \end{array}
 \end{array}
 \quad (7.4)$$

The calculations may then proceed as follows:

1. Calculate region (iii):

(a) If column LC is in (j), then:

$$z_3 = \tilde{A}_{11}^{-1} e^{LC} = (\tilde{A}_{11}^{-1})^{LC}$$

(b) If column LC is in (jj), then:

$$z_3 = -\tilde{A}_{11}^{-1} E_{11} (F_{11}^{-1})^{LC}$$

(c) If column LC is in (jjj), then:

$$z_3 = \tilde{A}_{11}^{-1} [(\tilde{E}_{13})^{LC} - E_{11} F_{11}^{-1} (F_{13})^{LC}]$$

notice that $(F_{11}^{-1})^{LC}$ and $F_{11}^{-1} (F_{13})^{LC}$ may be computed by backpath traversal on F_{11} (e.g., Bradley, Brown and Graves [1977], p. 11).

2. Calculate region (i):

(a) If column LC is in (j), then:

$$F_{11} z_1 = F_{12} z_3$$

(b) If column LC is in (jj), then:

$$F_{11} z_1 = F_{12} z_3 + e^{LC}$$

(c) If column LC is in (jjj), then:

$$F_{11} z_1 = F_{12} z_3 + (F_{13})^{LC}$$

3. Calculate region (ii):

(a) If column LC is in (j), then:

$$z_2 = -F_{22}z_3 - F_{21}z_1$$

(b) If column LC is in (jj), then:

$$z_2 = -F_{22}z_3 - F_{21}z_1$$

(c) If column LC is in (jjj), then:

$$z_2 = -F_{22}z_3 - F_{21}z_1 + (F_{23})^{LC}$$

4. Calculate region (iv):

(a) If column LC is in (j), then:

$$z_4 = -E_{22}z_3 - E_{21}z_1$$

(b) If column LC is in (jj), then:

$$z_4 = -E_{22}z_3 - E_{21}z_1$$

(c) If column LC is in (jjj), then:

$$z_4 = -E_{22}z_3 - E_{21}z_1 + (E_{23})^{LC}$$

Then column LC of (7.4) is available as $z^T = (z_1, z_2, z_3, z_4)^T$. Note that floating point multiplications and/or divisions are necessary only for computation of regions (iii) and (iv), and that regions (i), (ii) and (iv) are linear combinations of region (iii) and one another.

D. SEQUENCING COMPUTATIONS TO EXPLOIT COMMON SUB-EXPRESSION IN ROW GENERATION

Now we examine the actions of **Generate_Row**(LR). Suppose we desire to generate row LR and place the results of the computation in the vector $\tilde{z} = (\tilde{z}_5, \tilde{z}_6, \tilde{z}_7)$ which is partitioned to conform to (7.3). Rewriting (7.3) in a convenient form:

$$\begin{array}{l}
 \text{(i)} \quad \left(\begin{array}{ccc} \overbrace{-F_{11}^{-1} F_{12} \tilde{A}_{11}^{-1}}^{(i)} & -\left\{ \left[-F_{11}^{-1} F_{12} \tilde{A}_{11}^{-1} \right] E_{11} - I \right\} F_{11}^{-1} & \left[-F_{11}^{-1} F_{12} \tilde{A}_{11}^{-1} \right] E_{13} + \left\{ -\left[-F_{11}^{-1} F_{12} \tilde{A}_{11}^{-1} \right] E_{11} - I \right\} F_{11}^{-1} F_{13} \\ \text{(ii)} \quad \left(F_{21} F_{11}^{-1} F_{12} - F_{22} \right) \tilde{A}_{11}^{-1} & -\left\{ \left[\left(F_{21} F_{11}^{-1} F_{12} - F_{22} \right) \tilde{A}_{11}^{-1} \right] E_{11} + F_{21} \right\} F_{11}^{-1} & \left[\left(F_{21} F_{11}^{-1} F_{12} - F_{22} \right) \tilde{A}_{11}^{-1} \right] E_{13} \\ \text{(iii)} \quad \tilde{A}_{11}^{-1} & -\left[\tilde{A}_{11}^{-1} \right] E_{11} F_{11}^{-1} & + \left\{ -\left[\left(F_{21} F_{11}^{-1} F_{12} - F_{22} \right) \tilde{A}_{11}^{-1} \right] E_{11} + F_{21} \right\} F_{11}^{-1} F_{13} + F_{23} \\ \text{(iv)} \quad \left(E_{21} F_{11}^{-1} F_{12} - E_{22} \right) \tilde{A}_{11}^{-1} & -\left\{ \left[\left(E_{21} F_{11}^{-1} F_{12} - E_{22} \right) \tilde{A}_{11}^{-1} \right] E_{11} + E_{21} \right\} F_{11}^{-1} & \left[\left(E_{21} F_{11}^{-1} F_{12} - E_{22} \right) \tilde{A}_{11}^{-1} \right] E_{13} \\ & & -\left\{ \left[\left(E_{21} F_{11}^{-1} F_{12} - E_{22} \right) \tilde{A}_{11}^{-1} \right] E_{11} + E_{21} \right\} F_{11}^{-1} F_{13} + E_{23} \end{array} \right)
 \end{array}
 \tag{7.5}$$

Then we may proceed as follows:

1. Calculate region (j):

(a) If row LR is in (i), then:

$$\tilde{z}_5 = -(F_{11}^{-1})_{LR} F_{12} (\tilde{A}_{11}^{-1})$$

(b) If row LR is in (ii), then:

$$\tilde{z}_5 = (F_{21} F_{11}^{-1} F_{12} - F_{22})_{LR} \tilde{A}_{11}^{-1}$$

(c) If row LR is in (iii), then:

$$\tilde{z}_5 = (\tilde{A}_{11}^{-1})_{LR}$$

(d) If row LR is in (iv), then:

$$\tilde{z}_5 = (E_{21}F_{11}^{-1}F_{12} - E_{22})_{LR}\tilde{A}_{11}^{-1}$$

2. Calculate region (jj):

(a) If row LR is in (i), then:

$$\tilde{z}_6F_{11} = -\tilde{z}_5E_{11} - \epsilon_{LR}$$

(b) If row LR is in (ii), then:

$$\tilde{z}_6F_{11} = -\tilde{z}_5E_{11} - (E_{21})_{LR}$$

(c) If row LR is in (iii), then:

$$\tilde{z}_6F_{11} = -\tilde{z}_5E_{11}$$

(d) If row LR is in (iv), then:

$$\tilde{z}_6F_{11} = -\tilde{z}_5E_{11} - (E_{21})_{LR}$$

3. Calculate region (jjj):

(a) If row LR is in (i), then:

$$\tilde{z}_7 = \tilde{z}_5E_{13} + \tilde{z}_6F_{13}$$

(b) If row LR is in (ii), then:

$$\tilde{z}_7 = \tilde{z}_5E_{13} + \tilde{z}_6F_{13} + (F_{23})_{LR}$$

(c) If row LR is in (iii), then:

$$\tilde{z}_7 = \tilde{z}_5 E_{13} + \tilde{z}_6 F_{13}$$

(d) If row LR is in (iv), then:

$$\tilde{z}_7 = \tilde{z}_5 E_{13} + \tilde{z}_6 F_{13} + (E_{23})_{LR}$$

The row LR of (7.5) is available as $\tilde{z} = (\tilde{z}_5, \tilde{z}_6, \tilde{z}_7)$. Note that floating point multiplications and/or divisions are avoided for terms involving F_{12} , $F_{22}F_{13}$ and F_{23} , and that regions (jj) and (jjj) are linear combinations of region (j).

E. DATA STRUCTURES

Several suites of data structures have been proposed for algorithms designed to solve (PNE) (Johnson [1966], Srinivasan and Thompson [1973], Glover, Karney, Klingman and Napier [1974], Bradley, Brown and Graves [1977]). Our implementation is based on the last.

The purpose of the data structures is to define a graph that represents F_{11} . Such a graph forms a rooted spanning tree defined over the nodes of N_B , which we denote as $\mathcal{G}_{F_{11}}$. Note that because F_{11} is nonsingular, F_{11} must contain at least one singleton column. Since F_{11} is maintained in triangulated form, we associate with each row IR the column JC in which the element appearing on the diagonal of IR occurs. This association is analogous to the GUB row/key variable association defined for the GUB algorithm. We represent this association with the array KEY(), of type INTEGER and dimensioned from 1 to $(m + n)$, which records for each row of F_{11} the extrinsic index of the corresponding key variable, and for each column of F_{11} the extrinsic index of the corresponding basic factored row. KEY() is undefined for other row and column indices.

We require knowledge of the row ordering of F_{11} to define a triangulation. $PO()$ is an INTEGER array, dimensioned from 1 to p , which for each row IR of F_{11} records the extrinsic index of the row that follows IR in the current triangulation. This ordering is referred to as preorder, and the successor of IR is called its preorder successor. $PO()$ is undefined for factored rows not currently in F_{11} . Note that the triangulated column ordering of F_{11} may be deduced from $PO()$ and $KEY()$.

$PO()$ and $KEY()$ capture the triangulated row and column ordering information for a representation of F_{11} but provide no means for interpreting this representation as a rooted spanning tree. The predecessor function $p()$ is a well known method for representing trees and thus rooted spanning trees. To define $p(i_k)$, we associate with each node i_k of $\mathcal{G}_{F_{11}}$ the row index of the offdiagonal element in the k^{th} column of F_{11} , when the rows are ordered to correspond to a triangulation of F_{11} . Graphically, $p(i)$ may be iterated recursively to trace the backpath from node i to the distinguished root node of $\mathcal{G}_{F_{11}}$.

We represent $p()$ by the INTEGER array $P()$, dimensioned from 1 to p , which for each row IR in F_{11} records the extrinsic index of the predecessor of IR . It is convenient to keep a record of the sign of the diagonal element of each row of F_{11} . We use the sign bit of $P()$ to do so. Assume node IRP is the predecessor of node IR in the current representation of F_{11} . If the diagonal element in row IR is a plus one, then $P(IR) = IRP$, while if the diagonal element in row IR is a minus one, $P(IR) = -IRP$. In the graph $\mathcal{G}_{F_{11}}$, this may be interpreted as follows: if the actual orientation of arc $a^k = KEY(IR)$ is the same as that recorded in $\mathcal{G}_{F_{11}}$, which is (IR, IRP) , then $P(IR) > 0$. If the actual orientation of arc a^k is opposite that recorded in $\mathcal{G}_{F_{11}}$, then $P(IR) < 0$.

The final data structure, $D()$, is an INTEGER array dimensioned from 1 to p . For each node IR of $\mathcal{G}_{F_{11}}$ (row of F_{11}) $D(IR)$ records the depth of node IR , where

depth is defined as the number of nodes encountered on the backpath between IR and the root node. $D()$ allows updates to be performed on the data structures representing $\mathcal{G}_{F_{11}}$ in a single pass through the data structures, and allows us to avoid unnecessary operations in the solution of linear systems to be discussed shortly.

F. SOLVING LINEAR SYSTEMS

It is clear from the discussion of **Generate_Row**(LR) and **Generate_Column**(LC) that the solution of the linear systems,

$$z_1^T F_{11} = b_1^T$$

and

$$F_{11} z_2 = b_2$$

where z_1 and z_2 are vectors of unknowns and b_1 and b_2 are vectors of rationals, are crucial steps. The practical value of our implementation depends to a large extent on the efficiency with which these systems may be solved, and thus we will consider this problem in some detail.

First, we examine the data structures used to represent the terms. Three data structures are used to represent b_1 and b_2 , the right-hand side terms. **WORK()** is a **DOUBLE PRECISION** array, dimensioned from 1 to p , which is used to hold the real values of b_1 and b_2 (we sequence operations so that at any given moment we are interested in either b_1 or b_2 but not both, so that the same array may be used for both). An **INTEGER** array **WORKMK()**, also dimensioned from 1 to p , is used as a nonzero mask for **WORK()**. The convention is:

$$\begin{aligned} \text{WORKMK}(K) &= 0 & \text{if } \text{WORK}(K) &= 0.0 \\ \text{WORKMK}(K) &= 1 & \text{IF } \text{WORK}(K) &\neq 0.0 \end{aligned}$$

Finally, the array $\text{WORKNZ}()$, of type INTEGER and also dimensioned from 1 to p , records the extrinsic column (in b_1) or row (in b_2) index of each nonzero in $\text{WORK}()$. The use of $\text{WORKMK}()$ and $\text{WORKNZ}()$ allows the efficient management of nonzeros in b_1 and b_2 .

The solution vectors z_1 and z_2 are represented by three analogous arrays. $\text{ROWCOL}()$ (for rowcolumn()) is of type DOUBLE PRECISION and is dimensioned from 1 to $(m + n)$. It is used to store the representation of a row and column of the principal part of the tableau, and thus portions of it are used to store the solutions of Equation (7.1). Associated with $\text{ROWCOL}()$ is $\text{ROWCOLMK}()$, an INTEGER array of conformable dimension, used as a nonzero mask for $\text{ROWCOL}()$, and the INTEGER array $\text{ROWCOLNZ}()$, also of conformable dimension, used to record the indices of nonzeros in $\text{ROWCOL}()$.

In contrast to the usual situation in Simplex-based approaches to linear programming in which each successive right-hand side of the problem may be computed by means of a simple update to the previous right-hand side, in this setting such is not the case. Thus, we are not able to derive the solution to the current form of Equation (7.1) by simply updating the previous solution. Instead, we must solve each system from scratch.

Each right-hand side is itself the result of a sequence of preliminary computations. As each new nonzero element is generated by these computations, its value is placed in $\text{WORK}()$, $\text{WORKMK}()$ is updated and the index of $\text{WORK}()$ in which

the nonzero appears in placed in the next available position in WORKNZ(). For example, suppose the first nonzero generated for WORK() is the value 6.5 in position 38. Then the arrays appear as follows:

$$\begin{array}{rcl} \text{WORK}(38) & = & 6.5 \\ \text{WORKMK}(38) & = & 1.0 \\ \text{WORKNZ}(1) & = & 38.0 \end{array}$$

A counter for the number of nonzeros in WORK() is maintained (an INTEGER variable called NNZW), so at the conclusion of the preliminary computations we may iterate through the nonzeros on the right-hand side in the order in which they were generated by:

$$\text{WORK}(\text{WORKNZ}(K)), K = 1, \dots, \text{NNZW}$$

Since our representation of F_{11} is in upper triangular form, the obvious approach for solving

$$F_{11}z_2 = b_2 \tag{7.6}$$

is by back substitution. All nonzero elements in F_{11} are either plus ones or minus ones, so only assignments, additions and subtractions are necessary. Assuming that the current dimension of F_{11} is k by k , we proceed by considering each row IR in turn, $\text{IR} = k, \dots, 1$. We assign the value in $\text{WORK}(\text{IR})$ to $\text{ROWCOL}(\text{KEY}(\text{IR}))$ and update by an addition the value in $\text{WORK}()$ in the row corresponding to the predecessor of row IR. (i.e., $\text{WORK}(\text{P}(\text{IR}))$).

This approach requires knowledge of the ordering of the rows of F_{11} in the order of last to first in triangulated form, which is precisely the reverse of our data

$$\begin{array}{c}
100 \quad 101 \quad 102 \quad 103 \quad 104 \quad 105 \quad 106 \quad 107 \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \left(\begin{array}{cccccccc}
1 & 1 & -1 & & & & & \\
& -1 & & & & & & \\
& & 1 & & & & & \\
& & & -1 & 1 & & -1 & \\
& & & & -1 & 1 & & \\
& & & & & -1 & & \\
& & & & & & 1 & \\
& & & & & & & -1
\end{array} \right)
\end{array}$$

Figure 7.1: A Triangulated F_{11}

structure $PO()$. To support this approach in our implementation, we could define and maintain an additional data structure $EO()$, recording the “endorder” of F_{11} . If we choose not to include an additional data structure, we could instead invert $PO()$ in situ whenever this reverse ordering is required, inverting it again when $PO()$ is next required. With either approach, it is convenient to make $PO()$ and $EO()$, if included, circular lists and to add a distinguished “artificial node”, say IRA , which is then used to locate both the first node (row) in preorder and the first node in endorder. To illustrate, Figure 7.1 displays a triangulated form of F_{11} with row and column labels.

Figure 7.2 presents the corresponding graph $\mathcal{G}_{F_{11}}$. By assuming the existence of an artificial node IRA , we in effect change our paradigm of $\mathcal{G}_{F_{11}}$ to that shown in Figure 7.3.

We may interpret the backsolve method as a labeling procedure on $\mathcal{G}_{F_{11}}$. In this context, each nonzero in the right-hand side is interpreted as a supply or demand at the corresponding node in $\mathcal{G}_{F_{11}}$. The problem of solving Equation (7.6) is then interpreted as one of finding a set of feasible flows defined on the arcs of $\mathcal{G}_{F_{11}}$.

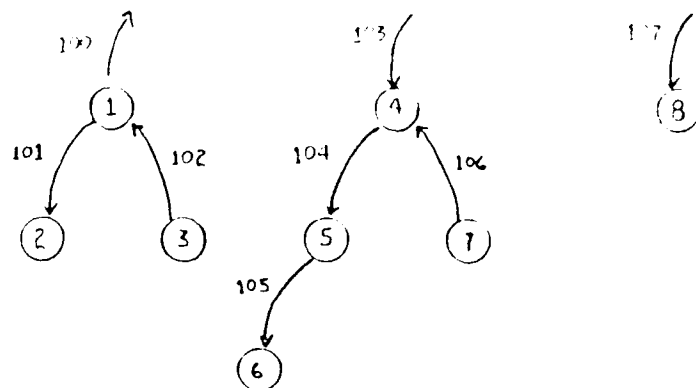


Figure 7.2: A Basis Graph $\mathcal{G}_{F_{11}}$

The backsolve approach to solving Equation (7.1) has the advantage of simplicity, and when viewed as a labeling algorithm has the intuitive appeal of “visiting” each node only once. The disadvantages are the need for either additional storage ($\text{EO}()$) or additional computation (two inversions of $\text{PO}()$).

The right-hand sides of these problems are invariably very sparse: the density is typically 0.3% or less. We are thus motivated to explore alternate approaches to solving Equation (7.6). The solution approach just outlined requires the “visiting” of every node in $\mathcal{G}_{F_{11}}$. Although $\text{WORKMK}()$ allows us to perform a simple INTEGER comparison to determine if the current node has nonzero supply or demand, we have strong empirical evidence that suggests that other approaches are more efficient. It is important to keep in mind that this problem is very deeply nested within the algorithm structure, and must be solved tens of thousands of times for a typical

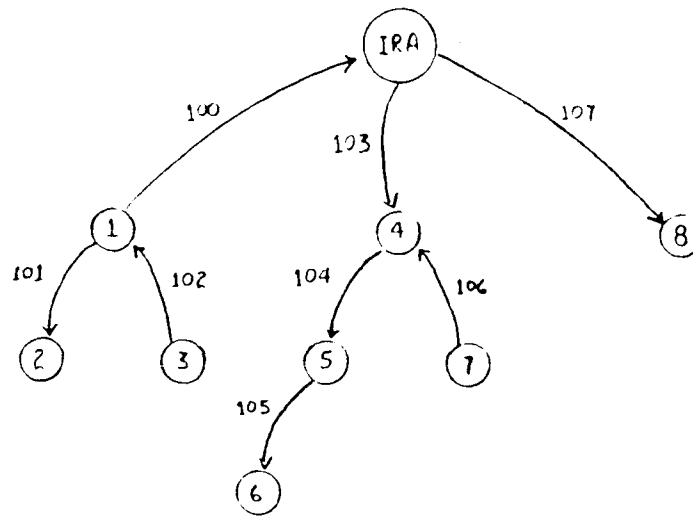


Figure 7.3: An Alternate Graph Paradigm $\hat{\mathcal{G}}_{F_{11}}$

linear programming problem. Small changes in efficiency here have large effects on the overall algorithm efficiency.

Since the right-hand sides are sparse, an alternative is to consider solving Equation (7.6) iteratively as a sequence of subproblems, each consisting of a right-hand side containing a single nonzero element and a current partial solution. Iterating through the nonzeros, we introduce the supply or demand at the corresponding node in $\mathcal{G}_{F_{11}}$, and then adjust the flows of all arcs appearing on the backpath of that node as necessary.

Analyzing the worst-case performance of these two approaches, we see that the first approach is linear in the number of nodes in $\mathcal{G}_{F_{11}}$ (rows in F_{11}), while the second approach is quadratic in the number of nonzeros in the right-hand side. For the densities typically encountered in the problems we have studied, the crossover point at which the performance of the linear algorithm overtakes that of the quadratic is

in the range of 400 to 1000 rows in F_{11} . However, in the testing we have performed, the worst-case analysis is pessimistic, and in practice the performance of the second approach always dominates that of the first.

The second linear system of interest is

$$z_1^T F_{11} = b_1^T . \quad (7.7)$$

We interpret the problem as one of being given flows on the arcs of $\mathcal{G}_{F_{11}}$ and being asked to find the necessary supplies and demands at the nodes. Analogies to each of the approaches to solving Equation (7.6) may be found for solving Equation (7.7), and our empirical evidence strongly suggests that an approach analogous to the second method for solving Equation (7.6) is preferable for solving Equation (7.7).

Having made the design decisions for solving Equation (7.6) and Equation (7.7), it is worthwhile to review our paradigm for $\mathcal{G}_{F_{11}}$. Our preferred solution techniques do not require a partial ordering of all rows of F_{11} . Rather, we require partial ordering information only among each set of coupled rows, or, graphically, among each disjoint component of $\mathcal{G}_{F_{11}}$. Rather than introducing an artificial node, we may treat each disjoint component as an independent entity. It is then convenient to treat singleton arcs (either dynamic or static singletons) as self-loops, and construct the predecessor function so that it forms a ring within each component. Our graph paradigm of Figure 7.1 then becomes as shown in Figure 7.4. The data structure representation of Figure 7.4 is then:

Node:		1	2	3	4	5	6	7	8
Predecessor:	P()	1	-1	1	-4	-4	-5	4	-8
Preorder Successor:	PO()	2	3	1	5	6	7	4	8
Depth:	D()	0	1	1	0	1	2	1	0
Key Variable:	KEY()	100	101	102	103	104	105	106	107

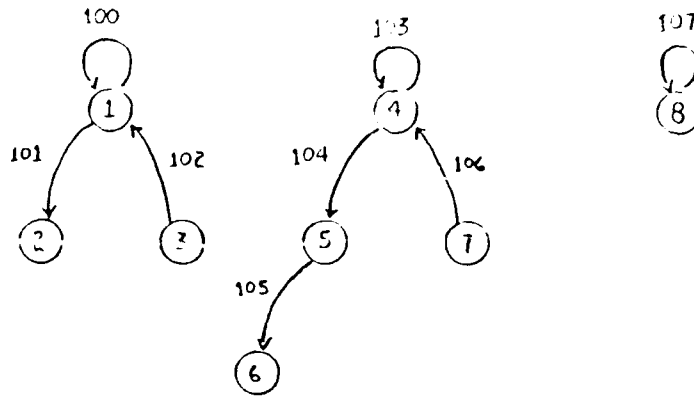


Figure 7.4: The Implementation Paradigm for $\mathcal{G}_{F_{11}}$

G. FACTORED KERNEL UPDATE

The update of a rooted spanning tree representation of a pure network triangulated basis which results from a column exchange is well understood and has been treated thoroughly in the literature. Bradley, Brown and Graves [1977] give an excellent account of such an update in an algorithmic setting very similar to the one here. Hence, we will not repeat the details. In this setting, however, a column exchange is but one of four possible updates required. We must also deal with row exchanges, row and column additions and row and column deletions. Our general approach will be to reduce these three cases to the column exchange case through a sequence of operations which may be thought of as pre- and post-processing. The following three cases, along with the column exchange case, comprise the actions required for `Update_Factored_Kernel`.

The first case we consider is the row and column deletion case, in which the dimension of F_{11} decreases. It is convenient to limit row/column deletions to row/key variable pairs. This is because removing a row/key variable pair always preserves

nonsingularity of the remaining factored kernel, and because, with our choice of data structures, given one member of the pair, it is easy to identify the other member through the use of the $KEY()$ array.

With our paradigm for $\mathcal{G}_{F_{11}}$, the key variable for row IR corresponds to the arc which connects node IR to its predecessor. The removal of IR and $KEY(IR)$ creates a new disjoint component within $\mathcal{G}_{F_{11}}$ consisting of all nodes and the associated key variables on whose backpaths IR and $KEY(IR)$ appear. If IR is a leaf (a node which is incident to a single arc), then the new disjoint component is null. Within the new disjoint component, $D()$ changes for all nodes, but $P()$ and $PO()$ change only for the first node in preorder, say IRFIRST, and the last node in preorder, say IRLAST. For the first node in preorder, its predecessor becomes itself (with the sign bit used to indicate arc orientation), forming a self-loop. The update is thus:

$$P(IRFIRST) = IRFIRST$$

Its depth is updated to be zero, and the magnitude of this depth change is recorded for future use. For every other node in the new disjoint component, $D()$ is reduced by the magnitude of the IRFIRST depth change. Finally, $PO()$ of IRLAST is changed to the first node in preorder, restoring the local ring structure of $PO()$. The update is:

$$PO(IRLAST) = IRFIRST$$

These updates can be accomplished in one pass through the data structures of the new disjoint component. The structure of $\mathcal{G}_{F_{11}}$ is restored by changing $PO()$ of the predecessor of IR to the index of the node which was the successor of IRLAST prior to the update.

The second update case is a row and column addition. Suppose IRADD is the row (node) to be added to F_{11} ($\mathcal{G}_{F_{11}}$). We wish to treat this update as a column exchange. To do so, we first incorporate IRADD into the structure and associate with it an imaginary ("logical") arc which forms a self-loop. We then perform a column exchange with the new column, say JCADD, entering $\mathcal{G}_{F_{11}}$ and the logical arc leaving $\mathcal{G}_{F_{11}}$.

The difficulty in incorporating IRADD into $\mathcal{G}_{F_{11}}$ lies with the dynamic singleton arcs. $\mathcal{G}_{F_{11}}$ currently may have many singleton arcs which appear in our paradigm as self-loops. If any of these dynamic singletons are actually incident to IRADD in \mathcal{G} , incorporating IRADD into $\mathcal{G}_{F_{11}}$ requires changing their representation.

To illustrate this point, assume IRADD has the node label "9" and we wish to incorporate it into the graph shown in Figure 7.4. Assume also that arcs 103 and 107 are dynamic singletons which are actually incident to node 9; that is, arc 103 = (9,4) and arc 107 = (9,8). We first initialize the data structures for node 9 as follows:

$$D(9) = 0$$

$$P(9) = 9$$

$$PO(9) = 9,$$

which has the effect of placing node 9 at depth 0 as a disjoint component consisting of a single node with a (logical) self-loop, as shown in Figure 7.5.

We then change the representation of arcs 103 and 107 from dynamic singletons to doubletons. To do this, we change $P(4)$ from -4 to -9 and $P(8)$ from -8 to -9. We then assert a partial ordering among those (formerly) disjoint components which have been merged by the change in the representation of the dynamic singletons.

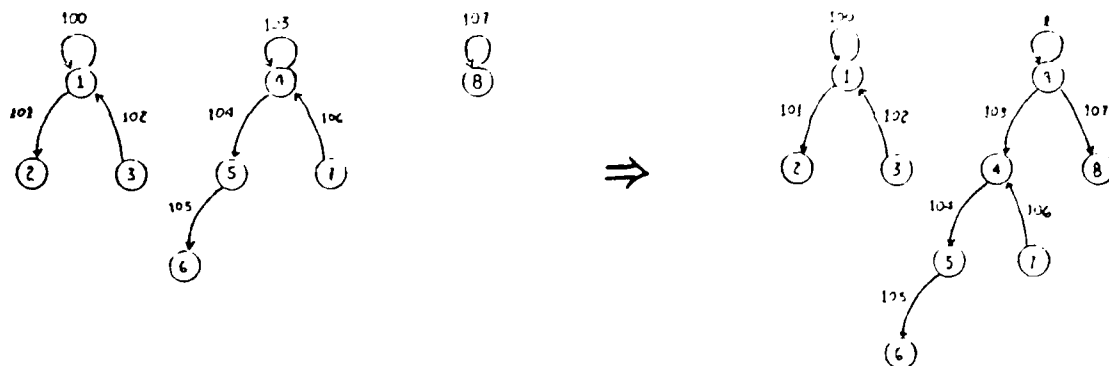


Figure 7.5: Initialization of Node 9 Prior to Incorporation into $\mathcal{G}_{F_{11}}$

and retriangulate. This is accomplished by increasing the depth of each node in the component by one, changing $PO(7)$ from 4 to 8, and $PO(8)$ from 8 to 9. This update requires a single pass through the data structures of each affected component. After incorporation, $\mathcal{G}_{F_{11}}$ appears as shown in Figure 7.6. We may now complete the update by performing a column exchange.

To identify the dynamic singletons which are affected by such an update, we access the extrinsic problem data structures for row (node) $IRADD$. For each column with a nonzero in row $IRADD$, we test whether or not that column is currently key for some row in F_{11} . If so, that column is currently represented in $\mathcal{G}_{F_{11}}$ as a dynamic singleton.

The final update case is the row exchange case. Suppose we want to replace row $IROUT$ with row $IRIN$. We treat this in two stages. The first stage is a row and column deletion case, in which the node $IROUT$ and the arc $KEY(IROUT)$ are

removed from $\mathcal{G}_{F_{11}}$. To complete the update, we seek a column (arc) which may be added to $\mathcal{G}_{F_{11}}$ along with row (node) IRIN. We require an arc which is either a static singleton, incident to node IRIN, or a doubleton, incident to node IRIN and a node (other than IROUT) which is currently in $\mathcal{G}_{F_{11}}$. We first consider arc KEY(IROUT). If it satisfies the second condition (obviously, it cannot satisfy the first condition), we designate it as the arc to be added. If not, we search among the variables (arcs) in region (iii) of the tableau for such an arc. We know one must exist, for otherwise B is singular. We select the first such arc found as the arc to be added. We then perform the row and column addition case.

Recall that the action **Is_Factored_Kernel_Singular()** is required when we have identified an arc (column) for removal from $\mathcal{G}_{F_{11}}$ and we are considering a candidate arc to replace it. We want to determine if this exchange preserves the nonsingularity of F_{11} . The fact that $\mathcal{G}_{F_{11}}$ is a rooted spanning tree provides the necessary structure to support a simple test of nonsingularity. Assume JCOUT is the (known) arc to be removed from $\mathcal{G}_{F_{11}}$ and JCIN is the candidate replacement. In our paradigm, F_{11} is nonsingular if and only if each disjoint component of $\mathcal{G}_{F_{11}}$ is connected and contains exactly one cycle, which must be a self loop occurring at the component's root (the root is the distinguished node in the component whose depth is zero). The removal of JCOUT creates a new disjoint component which has no root self loop. If the addition of JCIN fails to correct this, the proposed exchange is singular.

The specific test is this: identify the nodes incident to JCIN using the original problem data structures. Note that there may be zero, one or two such nodes. If none of these nodes are currently included in the structure of $\mathcal{G}_{F_{11}}(F_{11})$, then the proposed exchange is singular. For each such node which is currently included in the structure of $\mathcal{G}_{F_{11}}$, traverse the backpath of that node by recursively iterating

the predecessor array $P()$ until either the root node of that component is found, or until the "join" is located (the "join" is that node with largest depth which appears on the backpath of both nodes incident to JCIN. The join exists only if JCIN is incident to two nodes, both nodes are currently included in the structure of $\mathcal{G}_{F_{11}}$, and the two nodes are in the same disjoint component.). If JCIN is encountered during this backpath traversal, the proposed exchange is nonsingular. Otherwise, it is singular.

The action **Find_Column_to_Remove**(IOUT) is particularly simple, since, as mentioned previously, we always remove row/key variable pairs. Thus, if IR is the node to be removed, then $\text{KEY}(\text{IR})$ is the corresponding column to be removed.

Finally, the action **Find_Column_to_Add**(IIN) requires searching among the variables in region (iii) of the tableau. For each candidate arc, we invoke **Is_Factored_Kernel_Singular()**. If the response is "FALSE", we have found the column to be added. Otherwise, we continue the search.

VIII. FACTORIZATION OF GENERALIZED NETWORK ROWS

A. INTRODUCTION

The problem of interest remains:

$$\begin{aligned}
 & \min_y : \quad wy \\
 \text{(GNSC)} \quad & \text{s.t. :} \quad Fy \leq b; \quad F \text{ } p \text{ by } n \\
 & \quad Ey \leq r; \quad E \text{ } m \text{ by } n \\
 & \quad -Iy \leq 0; \quad -I \text{ } n \text{ by } n
 \end{aligned}$$

where F , E , $-I$, w , b and r are as before. We now require that each column of F have at most two nonzero elements, which may be of arbitrary sign. We may associate a generalized network with F by defining a node corresponding to each row i of F and an arc F^j corresponding to each column j of F . The arcs are defined as:

$$F^j = \begin{cases} (i, k) & \text{if } F_{ij} \neq 0, \quad F_{kj} \neq 0 \text{ and } i < k \leq p \\ (i, 0) & \text{if } F_{ij} \neq 0, \quad F_{kj} = 0 \text{ for all } k \neq i \leq p \\ (0, 0) & \text{if } F_{ij} = 0 \text{ for all } i \leq p \end{cases}$$

We let $\mathcal{A} = \{F^1, \dots, F^n\}$, $\mathcal{N} = \{1, \dots, p\}$ and define the graph $\mathcal{G} = \{\mathcal{N}, \mathcal{A}\}$. Arcs of the form $(i, 0)$ are singleton arcs, sometimes called root arcs. Arcs of the form $(0, 0)$ are null.

The most widely studied specialization of (GNSC) is obtained when the F -type constraints are equalities and the E -type constraints are vacuous, resulting in:

$$\begin{array}{ll}
 \min_y : & wy \\
 \text{(GNE)} & \\
 \text{s.t. :} & \begin{array}{ll}
 Fy \leq b; & F \text{ } p \text{ by } n \\
 Ey \leq r & E \text{ } m \text{ by } n \\
 -Iy \leq 0; & -I \text{ } n \text{ by } n
 \end{array}
 \end{array}$$

Dantzig [1963] provides the seminal treatment of (GNE), identifying the important structure that leads directly to efficient primal Simplex-based algorithms for its solution. Implementations have been reported by Glover, Klingman, Hultz, Karney and Elam [1972, 1973, 1977, 1978, 1979] and Brown and McBride [1984].

(GNE) may be viewed as a generalization of (PNE), and the cost of such generalization is a weakening of the properties which so strongly characterize (PNE) and which lead to the efficient and elegant implementations. In (GNE), F is not totally unimodular, and thus an implementation may not be restricted to integer arithmetic. It is not possible to triangulate every primal Simplex basis extracted from F by row and column permutations. The subgraph generated from the columns of a primal Simplex basis no longer form a rooted spanning tree defined on the nodes of \mathcal{G} . Apparently, much has been given up in the generalization.

In fact, significant structure remains in (GNE). A well known result, due to Dantzig [1963], shows that any primal Simplex basis extracted from F can be put in the form of Figure (8.1) by row and column permutations. Each square submatrix component B^k is either upper triangular or nearly upper triangular with only one element below the diagonal. Notice that if each B^k is strictly upper triangular, the structure is analogous to that found in the (PNE) primal Simplex basis.

Interpreting the structure of B^k as a subgraph of \mathcal{G} , we find that when B^k is upper triangular, the subgraph may be viewed as a disjoint component with a single self-loop at the root node, exactly as in the (PNE) case. When B^k is not

$$\begin{bmatrix} B^1 & & & & \\ & B^2 & & & \\ & & \ddots & & \\ & & & B^k & \\ & & & & \ddots \\ & & & & & B^q \end{bmatrix}$$

Figure 8.1: Near-Triangulated Simplex Basis Corresponding to (GNE)

upper triangular, the subgraph still forms a disjoint component with a single loop, but that loop is no longer a self-loop. For example, Figure (8.2) shows a nearly triangular B^k with row and column labels.

$$\begin{matrix} & \underbrace{100} & \underbrace{101} & \underbrace{102} & \underbrace{103} & \underbrace{104} & \underbrace{105} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{matrix} & \left(\begin{array}{cccccc} 1.08 & .98 & & & & \\ & 1 & .99 & & 1 & .98 \\ & & -1 & -1 & & \\ -1 & & & 1 & & \\ & & & & 1.01 & \\ & & & & & -1 \end{array} \right) \end{matrix}$$

Figure 8.2: A Sample Nearly-Triangulated (GNE) Simplex Basis Component

The resulting subgraph is shown in Figure (8.3). Such a subgraph is commonly called a “one-tree”, (an apparent oxymoron).

This structure allows the extension of the algorithm and data structures developed for (PNE), leading to efficient implementations to solve (GNE) (e.g., Brown and McBride [1984]).

(GNSC) has received less attention in the literature than (PNSC). Hultz and Klingman [1976] develop a primal Simplex-based approach to an equality-constrained formulation of (GNSC), and report on an implementation which allows

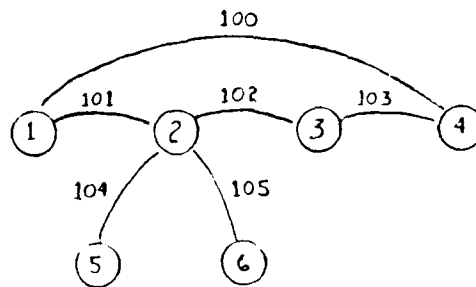


Figure 8.3: "One-tree" Subgraph

a single E -type constraint (Hultz and Klingman (1978)). McBride [1985] develops an algorithm for solving a generalization of (GNSC) in which complicating columns as well as complicating constraints are permitted, and reports on an implementation of that algorithm.

B. THE FACTORED TABLEAU

The algebraic development of the primal row basis B , the nonbasic rows D and the factored tableau is exactly as shown in Chapter 7 and is not repeated here. Note that F_{11} , the factored kernel, may now be placed in the form shown in Figure (8.1) by row and column permutations. The corresponding graph, $\mathcal{G}_{F_{11}}$, consists of one or more disjoint components, each of which contains exactly one loop. The loop may be either a self-loop, as in the case of the disjoint components of (PNSC), or

it may be a "root loop" (a loop consisting of two or more nodes, all of which are at depth zero in the component), as shown in Figure (8.3).

C. SEQUENCING COMPUTATIONS TO EXPLOIT COMMON SUB-EXPRESSIONS IN COLUMN GENERATION

The sequencing of computations for column generation in (GNSC) is exactly the same as that for (PNSC).

D. SEQUENCING COMPUTATIONS TO EXPLOIT COMMON SUB-EXPRESSIONS IN ROW GENERATION

The sequencing of computations for row generation in (GNSC) is exactly the same as that for (PNSC).

E. DATA STRUCTURES

Several proposed data structures for algorithms tailored to solve (GNE) have been offered (Glover, Klingman and Stutz [1973, 1974], Elam, Glover and Klingman [1979], Brown and McBride [1984]). Our implementation is based on the last, which extends to (GNE) the data structures developed in Bradley, Brown and Graves [1977] to solve (PNE).

We have seen the similarity between the structure of the disjoint components that arise in the graph corresponding to F_{11} in (GNSC) and the structure of the components in (PNSC). To develop a representation of these components, we wish to extend the data structures developed for (PNSC) in a natural way. Knowledge of the row ordering of F_{11} is again maintained in an INTEGER array $PO()$, dimensioned from 1 to p . As before, it is convenient to make $PO()$ a circular list defined on each disjoint component. The unique correspondence between each row of F_{11} and a distinguished column is maintained in the INTEGER array $KEY()$, dimensioned

from 1 to $(m + n)$. The conventions for `KEY()` are exactly as in (PNSC). The depth array, $D()$, is defined exactly as before. However, as suggested by our diagram in Figure (8.3), we adopt the convention of placing all nodes appearing on a loop at depth zero, which is consistent with our definition of "root loop". Finally, a representation of the predecessor function is needed, and we define $P()$ as before. For any row IR in the factored kernel, $P(IR) > 0$ indicates that the diagonal element in the near-triangulation is the first non-zero factored element in its column.

F. SOLVING LINEAR SYSTEMS

Just as in (PNSC), the crucial steps in the generation of rows and columns of the principal part of the tableau are solving the systems

$$z_1^T F_{11} = b_1^T \quad (8.1)$$

and

$$F_{11} z_2 = b_2 \quad (8.2)$$

where z_1 and z_2 are vectors of unknowns and b_1 and b_2 are vectors of rationals. Our approach for solving these systems closely parallels that developed for (PNSC).

The data structures used to represent z_1, z_2, b_1 and b_2 are exactly the same as in the (PNSC) implementation. To represent b_1 and b_2 , we use:

`WORK()`

`WORKMK()`

`WORKNZ(),`

and for z_1 and z_2 :

ROWCOL()

ROWCOLMK()

ROWCOLNZ().

The right-hand side sparsity-exploiting approach for solving (8.1) and (8.2) developed in Chapter 7 may still be used, but two complications arise in the (GNSC) setting. First, the nonzero elements of F_{11} are no longer restricted to be plus and minus ones, and may assume arbitrary values. Thus, we are not able to restrict all arithmetic operations to additions and subtractions.

Interpreting (8.1) and (8.2) as problems of finding balancing supply and demands (8.1) or feasible flows (8.2), we see that in (GNSC) we must allow for gains and losses in the flows over the arcs of $\mathcal{G}_{F_{11}}$. Because our formulation allows two arbitrary nonzeros in each column of F (rather than one arbitrary nonzero and one plus one, for example), backward and forward substitution schemes require both multiplications and divisions. We may eliminate the need for divisions by pre-computing both ratios of the nonzero elements in each arc. That is, if a column has nonzero elements a and b in rows of F , we pre-compute the ratios $\frac{a}{b}$ and $\frac{b}{a}$. We then substitute the pre-computed value for the division whenever it occurs. Storing a pair of DOUBLE PRECISION real numbers for every column of (GNSC) is wasteful, and instead we define a single pointer for each column which points to the location of the first of two DOUBLE PRECISION real values. The first is the value $\frac{a}{b}$ and the second is $\frac{b}{a}$. (Note that for singleton arcs, the values a and $\frac{1}{a}$ are stored instead.) We compute and store only the unique ratio pairs for a given problem instance, and

the number of such unique pairs is usually quite small: typically, a problem with 10,000 to 20,000 columns has only 30 to 40 unique ratio pairs.

The second difficulty is that the disjoint components of F_{11} may be nearly triangular rather than triangular, and thus backward and forward substitution cannot be applied directly. However, Dantzig [1963] shows that a variation of backward and forward substitution may be used to solve (8.1) and (8.2). This method solves the triangulated part of a disjoint component exactly as is done in backward and forward substitution. When a root loop is encountered, the method requires two calculations for each row or column of F_{11} corresponding to a node or arc on the loop. The second calculation involves a term Dantzig [1963] called the "loop factor", which is common value for every node (or arc) on the loop. Our implementation uses this modified forward and backward solution technique for solving (8.1) and (8.2), and we store unique values of loop factors in a DOUBLE PRECISION array called GNROOT().

G. FACTORED KERNEL UPDATE

We first consider the actions required by **Update_Factored_Kernel**.

Brown and McBride [1984] give an excellent treatment of the update required for a column exchange within a generalized network basis in an algorithmic setting very similar to the one here, and thus we do not repeat that discussion. As in the case of (PNSC), however, three additional classes of updates may occur in this setting: row exchanges, column and row additions and column and row deletions. We again treat these three cases by reducing them to the column exchange case through sequences of pre- and post-processing operations.

We again limit row and column deletions to row/key variable pairs. When row IR and its associated key variable KEY(IR) are removed from F_{11} , a disjoint

component is created which has no loop. If IR and KEY(IR) do not appear on a root loop of $\mathcal{G}_{F_{11}}$, a new disjoint component is created, just as in the (PNSC) algorithm. If IR and KEY(IR) do appear on a root loop of $\mathcal{G}_{F_{11}}$, the resulting retriangulated component contains a self-loop rather than a root loop. The update of the subgraph data structures is very similar to those presented in Chapter 7.

Row and column additions require the incorporation of the incoming node into the structure of $\mathcal{G}_{F_{11}}$ prior to the column exchange, just as in (PNSC). The technique for (GNSC) is exactly the same as in (PNSC). Once the new node has been added to $\mathcal{G}_{F_{11}}$, a column exchange operation is performed, with a logical arc (which forms a self-loop on the new node being added to $\mathcal{G}_{F_{11}}$) being replaced by the new arc (column) being added.

Finally, row exchanges are handled as a two-stage update, exactly as in (PNSC).

The task of determining the singularity of F_{11} (**Is Factored Kernel Singular**) is more challenging in (GNSC) than in either of the other two algorithms due to the arbitrary nonzero elements. In each of the previous two algorithms, we have been able to determine with certainty the singularity of F_{11} indirectly. In the (GUB) factorization, $F_{11} = \Delta_{11}$, a signed identity matrix. Since Δ_{11} is orthogonal (i.e., $\Delta_{11}^T \cdot \Delta_{11} = I$), F_{11} is perfectly conditioned (see e.g., Golub and Van Loan [1983] for a discussion of matrix conditioning). In (PNSC), F_{11} may be triangulated, placing it in a form with plus and minus ones on the diagonal. Thus, the determinant of F_{11} is either plus or minus one. We may therefore determine its singularity by examining the structure of $\mathcal{G}_{F_{11}}$ rather than considering F_{11} directly. In either case, as long as the accumulated round-off error in the current representation of the problem is such that we can distinguish plus or minus one from zero, we may rely on the structure of our representation of $\mathcal{G}_{F_{11}}$ to deduce the singularity of F_{11} . Thus, we are able to discern singularity logically and need not resort to analytic methods.

In (GNSC), however, we may not rely exclusively on the structure of $\mathcal{G}_{F_{11}}$ to reach conclusions about the singularity of F_{11} . It is not difficult to invent examples of factored kernels corresponding to instances of (GNSC) which are ill-conditioned. We expect ill-conditioning of F_{11} to lead to serious numerical difficulties, which we wish to avoid if possible. Since we generally have freedom in determining the structure of F_{11} and since our fundamental rank-one update of F_{11} is the column exchange, we use a heuristic based on column exchanges to anticipate conditioning problems. Recall in the column exchange update, a column has been selected for removal from F_{11} and a candidate column has been identified as its replacement. We use the backpath traversal method described in Chapter 7 to determine if the proposed exchange maintains the required structure of $\mathcal{G}_{F_{11}}$. If it does not, we can conclude that the exchange renders F_{11} singular and we reject the candidate arc.

As we traverse the backpath, we perform calculations which, if the exchange does in fact preserve the required structure of $\mathcal{G}_{F_{11}}$, will ultimately compute the determinant of the disjoint component in which the replacement arc will appear if the proposed exchange is performed. If the absolute value of the computed determinant is too small (a decision controlled by a implementation parameter), we conclude that the proposed update produces a submatrix of F_{11} (the submatrix corresponding to the new disjoint component) that may be ill-conditioned, and thus F_{11} may itself be ill-conditioned. We therefore reject the proposed candidate arc.

This approach is attractive because it is computationally inexpensive and, based on our empirical evidence, it works well in practice. It is of course merely a heuristic, since it is easily shown (e.g., Golub and Van Loan [1983]) that the determinant can be a poor indicator of matrix condition. Further, we do not actually compute the determinant of F_{11} , but rather only the determinant of a submatrix of F_{11} .

The remaining update actions, **Find_Column_to_Add(ICI)** and **Find_Column_to_Remove(ICI)** are treated in a manner analogous to that in (PNSC).

IX. COMPUTATIONAL RESULTS

A. INTRODUCTION

The algorithms developed here have been implemented and extensively tested within the framework of a commercial-quality optimization system: the X-System of Brown and Graves [1975]. This system employs the Graves mutual primal-dual algorithm in a variety of large scale optimization applications, including linear, non-linear, mixed integer and decomposed models. Although we report computational results only for linear models, our factorizations are seamlessly integrated into the X-System and support all system features.

B. TEST PROBLEMS

The benchmark test suite is drawn from a wide variety of actual applications. Table (9.1) provides a short synopsis of each model, quoting from the abstract where a reference in the open literature is available. In some cases, several different instances of models are reported. We selected these models because they provide a representative sample of size, structure and taxonomy of contemporary real-life applications of linear programming. For those models that are mixed integer, we report solution statistics for the initial linear programming relaxation.

TABLE 9.1: Description of Test Suite Models

- **GTE** "The seven Telephone Operating Companies within GTE have adopted an integrated business system called Capital Program Management System (CPMS) to guide their 3 billion dollar per year capital planning. The system includes a large scale mixed integer programming optimization system that

optimizes the critical economic tradeoffs between maximizing the long-term budget value of the firm's equity and satisfying shorter-term financial constraints, resource limitations and service objectives. Investment opportunities for the next 5 years are modeled as 0-1 variables with alternative implementations for each. The objective is to maximize the net present value of the capital portfolio. There are financial constraints on capital, internally generated funds, net income to common, and limits on resources such as labor hours, lines installed, etc. There are also constraints that enforce logical relationships among opportunities (such as, if choose A then must choose B)." See Bradley [1986].

- **INVEST** "Capital allocation and project selection for a large multi-national firm is modeled as a two-stage multi-year nonlinear capital budgeting problem with over 40,000 integer variables. Innovative modeling yields subproblems easy to solve, and optimality is achieved with a single iteration of the nonlinear master problem." See Harrison, Bradley and Brown [1989]. The instance reported here is a linear program subproblem of the two-stage model.
- **TANKER** "A crude oil tanker scheduling problem faced by a major oil company is solved using an elastic set partitioning model. The model takes into account all fleet cost components, including the opportunity cost of ship time, port and canal charges, demurrage and bunker fuel. The model determines optimal speeds for the ships and the best routing of ballast (empty) legs, as well as which cargoes to load on controlled ships and which to spot charter. All feasible schedules are generated, the cost of each is accurately determined and the best set of schedules is selected." See Brown, Graves and Ronen [1987].

- **GAS** "Natural gas utilities supply about one fourth of the energy needs of the United States. From wellhead to consumer, operations are governed by an astounding diversity of purchase, transport and storage contract agreements which enable complex physical distribution systems to meet future demands no more predictable than next year's weather. Gas is a highly detailed optimization model which utilities use to plan operations and to justify such plans to regulatory agencies is developed." See Avery, Brown, Rosenkranz and Wood [1989].
- **ODSM** "A commonly occurring problem in distribution system design is the optimal location of intermediate distribution facilities between plants and customers. A multicommodity capacitated single-period version of this problem is formulated as a mixed integer linear program. A solution technique based on Benders Decomposition is developed. ... An essentially optimal solution is found and proven with a surprisingly small number of Benders cuts." See Geoffrion and Graves [1974]. The instances reported here are decomposition master problems.
- **TAM** "The annual decision on how much of the Air Force procurement budget should be spent on the many different aircraft and how much should be spent on the many different munitions is of great interest to many people. How the Air Force staff develops information to support the decision has changed over the years. Currently, a linear program is being used by the Air Force Center for Studies and Analysis and is being tested by the Munitions Division of the Plans and Operations Directorate (AF/XOXFM) for munitions tradeoff analysis. The LP uses existing data and estimates on (1) aircraft and munition effectiveness, (2) target value, (3) attrition, (4) aircraft and munition costs,

and (5) existing inventories of aircraft and munitions. Other factors considered are weather and length of the conflict." See Might [1987] and Jackson [1989].

- **PHOENIX** "The U. S. Army operates a fleet of over 7,000 helicopters to perform combat and combat support tasks. Although newer, more technically advanced helicopters have been and are being procured, the majority of the fleet is still composed of helicopters that were built during the late 1960s for use in South Vietnam. These older airframes are rapidly reaching the end of their useful lives and must be (i) replaced by newer, more advanced designs, (ii) gutted and refit or replaced by a combination of (i) and (ii). Army force planners recognize that this problem can only be solved by a long-term program and the commitment of billions of dollars. Phoenix is a software system employing mixed integer linear programming to help Army aviation staff and officers develop long- range helicopter modernization plans." See Clemence, Teufert, Brown and Wood [1988].
- **AMMO4H** "A four-commodity transshipment model for delivery over time of military products from production and storage locations to overseas locations to support theater operations is developed. The model covers five physical echelons, including production plants, storage depots, ports of embarkation, ports of debarkation and geographic field locations. Road, rail, sea and air transportation are modeled, and product demands are time- phased. Capacitation occurs primarily on sea and air links, and as throughput capacities on transfer points, requiring replication of some echelons. The objective of the model is to minimize deviation from on-time deliveries." See Staniec [1984].
- **GK** A weekly multi-plant production/inventory/transshipment linear program from a consumer products industry is developed. The model is meant

to guide weekly processing and packaging decisions. Production consists of two stages: basic products are produced and then packaged into different-sized containers to yield finished products. Processing lines typically produce a subset of the basic products and have limited capacity with overtime charges for weekend shifts. Packaging lines for finished products are similar. In-house inventory capacity is limited although outside storage is available at additional cost. Inter-plant shipments are made by rail or truck. See Wood [1989].

C. METHODOLOGY

We wish to evaluate implementations of our three algorithms on the basis of computation time and computer memory requirements. Since each algorithm is simplex-based, the formal theory of algorithmic complexity provides no basis for preferring one to another, since in the worst case none enjoys a measure of running time that is polynomial in the size of the problem specification (see e.g., Garey and Johnson, [1979] for a discussion of algorithmic complexity and Klee and Minty [1972] for an analysis of the Simplex method). Thus, we are led to consider "typical" performance by gathering empirical evidence on the performance of the algorithms solving "typical" problems.

We prefer an implementation that is both fast and requires little computer memory. Most researchers who have reported on implementations of related algorithms have been concerned primarily with execution speed, and certainly it is important. However, we have seen in our algorithmic setting that once high speed storage has been allocated for the problem representation and for the program code, all remaining memory is available to store the representation of the explicit transformation kernel and the factored kernel. If the solution trajectory is such that their combined size never exceeds available memory, we succeed in solving the problem

If not, we fail. When success or failure depends on the total storage requirements of the inverse representation, we may be willing to sacrifice execution speed in exchange for an economical representation of the inverse. This is a classic theme in computer science and software engineering, and we believe its importance in this context has been largely overlooked.

We will be comparing the performance of four separate implementations. The first is an unadorned version of the X-System, which implements the Graves mutual primal-dual method as presented in Chapter 2. There is no underlying factorization. We refer to this implementation as "X". The second implementation is the GUB factorization presented in Chapter 6, and referred to as "GUB". The third is the pure network factorization of Chapter 7, referred to as "PNET", and the last is the generalized network factorization, called "GNET", of Chapter 8.

The ideal approach for this computational study, would be to develop four equivalent formulations of each model, each customized for its particular implementation with the goal of inducing a large factored row set of the appropriate type. This approach is a consistent theme in the literature dealing with specialized algorithms and one that we strongly endorse. Alternate formulations of a model are often available, and it seems sensible to choose one that as strongly as possible exploits the strengths of the solver.

However, all of the models used here are "off-the-shelf" in the sense that they were developed at various times by various modelers, and we cannot afford to develop alternate formulations. Thus, our approach is to preserve a single, unfactored representation of each model, and attempt to identify favorable row structures through the use of heuristics. Our procedure is based on the work of Brown and Thomen [1980], Brown and Wright [1983] and Brown, McBride and Wood [1985]. The heuristics are greedy and myopic in the sense that they initially consider the entire row

set of the problem, and discard one row at a time without backtracking until the remaining set satisfies the desired row factorization. This may easily result in the confounding or destruction of structure intended or perceived by the modeler. While our approach yields interesting and useful observations about the implementations, it is in some ways a poor substitute for the customized model method.

Table (9.2) tabulates the important structural information concerning the model instances we will be solving. The column headings may be interpreted as follows: m is the total number of structural constraints (note that this use is different from that in the problem specifications of Chapters 6, 7, and 8), n the number of variables, p_{GUB} the number of GUB rows identified by the heuristic, p_{PN} the number of pure network rows, p_{GN} the number of generalized network rows and NZ the number of nonzeros in the technological coefficient matrix of the model. For example, consider the first model in Table (9.2), GTE. The structural constraints contain 57,563 nonzeros, and the model consists of 6,624 variables. When viewed as an unfactored mutual primal-dual model, it consists of 960 explicit constraints and 0 factored constraints. When viewed as a GUB factorization, it consists of 909 factored (GUB) constraints and $960 - 909 = 51$ explicit constraints. Similarly, when viewed as a pure network factorization (PNET), it contains 909 factored rows and 51 explicit rows. Finally, when viewed as a generalized network factorization (GNET), it consists of 922 factored rows and 38 explicit rows.

D. COMPUTATIONAL RESULTS

We solved each of these problem instances on an IBM 3033/AP under the VM/CMS operating system using VS FORTRAN 1.4.1. A virtual machine size of six megabytes was used, simply because it is the largest size normally available to us. It is the nature of a time-shared system that measurements of processing

times are somewhat imprecise due to system load factors and accounting techniques for system processing overhead. We have attempted to mitigate these effects by performing our experiments during periods of low system usage.

Table (9.3) displays the solution times for each of the test problem instances by each of the four implementations. An “*” indicates that the solver failed to solve the problem instance. Such a failure occurred because the storage requirements for the explicit transformation kernel representation exceeded the memory available and the solver terminated gracefully. Solution times represent only the CPU time required to solve the problem, and exclude the initial problem input, the time required by the factorization heuristic to identify the factored row structure and the final output to record the solution. All figures are in CPU seconds.

The formulations of three of the test problems were strongly influenced by the design of the target solver: the TANKER model possesses a strong GUB structure since it contains a set of *schedule selection constraints* for each ship (i.e., from a set of candidate schedules, select at most one). The AMMO4H model is a multicommodity capacitated transshipment problem and is thus best suited to a pure network factorization. The PHOENIX10 model design was shaped by the generalized network factorization paradigm. The nature of the factored row structures shown in Table (9.2) supports this assertion. In the TANKER model, the factored pure network rows are exactly the same as the the factored GUB rows, and the heuristic constructs a generalized network factored row set by identifying one additional row to be paired with each GUB row. The AMMO4H model may be viewed as a GUB factorization with a relatively modest GUB set consisting of the joint capacitation constraints, or as a pure network factorization with a relatively large pure network (PN) factored row set. In the PHOENIX10 model, the dominant structure is clearly the generalized network row structure.

As we expect, the factorization is most successful when the model is wed to the solver. Although we are surprised to find the performance of (PNET) competitive with (GUB) on the TANKER model, (GUB) clearly dominates the X and GNET solvers. Similarly, (GNET) dominates on the PHOENIX10 model and (PNET) on the AMMO4H model. We would be disappointed if the results were otherwise.

We see evidence in Table (9.3) to suggest that the approach of using heuristics to automatically identify factored structure has its pitfalls. In a number of problem instances, although our heuristics identify significantly larger factored sets as we progress from the base system to (GNET), we see little improvement in computation times (INVEST, ODSM1, TAMS). In fact, we see in the TANKER model that the temptation to confound the modeler's intended GUB structure by specifying a generalized network factorization leads to disastrous consequences, even though this tactic doubles the size of the factored row set. Interestingly enough, when we apply the (GNET) solver to the (GUB) factored row set, we are able to solve the problem in 16.5 CPU seconds. This suggests that the "quality" of a row factorization is not completely specified by size alone.

We are encouraged by the observation that the transition from the basic system to (GUB) to (PNET) seldom degrades solution times, even when doing so yields little gain in the number of additional factored rows. This seems to contradict popular folklore, which suggests that computation times worsen as the transition from unadorned Simplex to (GUB) to (PNET) is made unless the transition is accompanied by a substantial increase in the size of the factored portion of the model. In fact, computational testing reported by others is frequently limited to models in which the number of explicit rows is in the range of one to twenty (see e.g., Chen and Saigal [1977], Glover, Karney, Klingman and Russell [1978], Glover and Klingman [1981]). Our results are all the more remarkable given the the lack of

guidance from the modeler for the "intended" row factorization. We note, however, that results are mixed for the transition from (PNET) to (GNET), and it seems clear that the applicability of the (GNET) factorization is not as general as that of (PNET).

Finally, we observe that in several models it is factorization that separates success from failure in solving the problem with a given allocation of computer resources. This fact alone may be reason to consider this approach in practice.

Our second interest with respect to computation is in the memory requirements of our algorithms. Our design strategy allocates memory to the data structures which represent F_{11} so that we may successfully represent the largest dimension of factored kernel that may possibly arise. Limited memory remains to store the representation of \tilde{A}_{11}^{-1} , the explicit transformation kernel. During the solution process, if we encounter a representation of \tilde{A}_{11}^{-1} which requires more memory than is available, failure occurs. We measure the size of the computer representation of \tilde{A}_{11}^{-1} and the amount of available computer storage in terms of the elements of \tilde{A}_{11}^{-1} that can be stored. The number of bytes per element varies according to the size of the problem (this has to do with the FORTRAN data types INTEGER*2 and INTEGER*4), but is generally 28 bytes per element. Table (9.1) lists for each problem instance/solver pair the maximum size of the \tilde{A}_{11}^{-1} representation encountered during the solution, measured in units of number of elements. An asterisk (*) indicates that the number shown equals the maximum number of units of storage that were available, and thus failure occurred.

We see that generally the representation of the maximum size of the explicit transformation kernel decreases as the generality of the factorization increases. Recalling the definition of the explicit transformation kernel:

$$\tilde{A}_{11}^{-1} = (E_{12} - E_{11}F_{11}^{-1}F_{12})^{-1}$$

this trend is as we would expect. As the generality of the factorization increases, we expect the size of the factored component to increase and the size of the explicit component to decrease. Each potentially binding explicit row which may be converted to a factored row by adopting a more general factorization reduces the dimension of the resulting representation of \tilde{A}_{11}^{-1} . Also, the density of the term $-E_{11}F_{11}^{-1}F_{12}$ generally increases as the complexity of the structure of F_{11}^{-1} increases. Assuming the dimension of F_{11} is k by k , the number of nonzeros in F_{11}^{-1} in the GUB factorization is k . In the pure network factorization, the number of nonzeros in F_{11}^{-1} may be as large as $\frac{n^2}{2}$, and in the generalized network factorization, the number of nonzeros in F_{11}^{-1} may be as large as n^2 . We note that (GNET) again provides several exceptions to the general trend in Table (9.4).

It is the dynamic nature of our factorization algorithms which marks this work as a departure from previous research. Table 9.5 illustrates the significance of this point. The first column lists the number of constraints which are binding at optimality, and the second column expresses this as a percentage of the total number of constraints in the problem instance. The results shown here are typical of real-world large-scale models. It is usually the case that many constraints are not binding at optimality, and there are computational advantages to be gained by exploiting this fact.

Columns 3, 4 and 5 of Table 9.5 list for (GUB), (PNET) and (GNET) respectively the number of explicit constraints that are binding at optimality. Since in each implementation, binding factored constraints are handled more efficiently than binding explicit constraints, we see that the computational success of our dynamic

factorization algorithms is due to the fact that even in large model instances, we are able to limit our attention to a relatively small number of explicit constraints, usually on the order of a few hundred or less. While this is well beyond the size of previously reported implementations, our results show that it is quite manageable.

It is useful to establish a criteria for comparing and contrasting the performance of the algorithms which accounts for both execution time and storage requirements. Although more sophisticated models could undoubtedly be developed, we offer a simple model which we feel captures the essential features we wish to consider. Define

$$f(s, t) = s \cdot t$$

where s is the total computer storage (measured in megabytes) required to solve a problem instance (including program code, original problem data, tableau representation, factored kernel representation and explicit transformation kernel representation) and t is the execution time (measured in CPU seconds). $f(s, t)$ is monotonically increasing in s and t , and in a crude fashion it captures the essential features of the way in which computer resources are often marketed commercially. We will use $f(s, t)$ as a measure of performance. Table (9.6) displays $f(s, t)$ for each of the problem instance/solver combinations. “*” indicates that the solver failed to solve the particular instance.

We observe that the trend as the transitions are made from base system to (PNET) is a decline in $f(s, t)$. It is apparent that (PNET) is a versatile implementation, performing extremely well on models with highly favorable structure (AMMO4II, KG4, GASPNA, GASPNC) and comparing favorably with (GUB) and

(GNET) on every other model. The performance of (GNET) is generally quite comparable to that of (PNET). It appears to be about 15-20% slower than (PNET) when it is used to solve a pure network factorization (GASPNC, AMMO4H). Our row elimination heuristics are usually effective in identifying a favorable factored structure, but the computational results on the TANKER model clearly illustrate the limitations of this approach. (GNET) apparently requires a more sophisticated and careful user than do (GUB) or (PNET), and in this sense it is perhaps a more specialized algorithm. The evidence shown here indicates that (PNET) is a strong candidate for use as a general implementation, and need not be viewed as a highly specialized implementation suitable only for rare model instances. For the models studied here, we have progressed well beyond the stage of solving instances with only a handful of explicit rows.

TABLE 9.2: Summary of Problem Suite Dimensions

	<i>m</i>	<i>n</i>	<i>p</i> _{GUB}	<i>p</i> _{PN}	<i>p</i> _{GN}	NZ
GTE	960	6,624	909	909	917	57,563
INVEST	1,338	11,989	941	1,101	1,168	36,829
TANKER	83	7,598	33	33	66	30,890
GAS PN A	6,848	27,884	4,345	5,934	5,976	36,702
GAS PN C	3,794	15,362	2,658	3,418	3,420	19,701
GAS PN E	1,184	5,102	434	877	883	7,355
GAS GN A	6,848	27,884	4,484	5,142	5,976	36,702
GAS GN C	3,974	15,362	2,664	3,084	3,420	19,701
GK 2	3,819	17,844	1,265	2,578	2,585	34,809
GK 3	5,728	27,493	2,295	3,867	3,876	54,289
GK 4	7,636	37,139	2,428	5,156	5,167	73,766
ODSM1	3,023	11,568	523	540	558	21,532
ODSM2	594	22,211	490	490	490	42,827
TAM1	91	389	22	28	34	1,212
TAM2	180	1,204	42	54	66	6,869
TAM3	269	2,883	63	81	99	21,356
TAM4	211	1,327	59	77	98	6,954
TAM5	438	10,969	102	132	162	93,964
TAM8	420	6,104	118	154	196	49,376
TAM12	629	17,793	177	231	295	164,947
PHOENIX10	1,618	6,884	206	220	1,153	13,818
PHOENIX30	4,305	4,297	293	303	3,605	16,441
AMMO 4H	13,963	83,497	1,071	12,892	12,892	166,713

TABLE 9.3: Solution Times in CPU Seconds

	X	GUB	PNET	GNET
GTE	48.4	33.0	37.2	38.3
INVEST	21.6	17.1	17.1	17.0
TANKER	141.8	20.5	17.7	43.8
GAS PN A	*	*	413.2	527.7
GAS PN C	56.9	53.7	35.8	31.3
GAS PN E	*	146.8	17.4	20.7
GAS GN A	*	*	*	521.6
GAS GN C	52.9	50.9	37.6	33.7
GK 2	27.1	22.8	21.3	19.4
GK 3	62.4	58.4	50.4	45.1
GK 4	*	380.9	182.2	186.6
ODSM1	8.7	12.1	9.3	8.8
ODSM2	34.1	32.5	32.8	30.4
TAM1	0.7	0.7	0.7	0.7
TAM2	2.5	1.7	2.2	2.1
TAM3	17.8	14.1	14.3	13.5
TAM4	1.2	1.2	1.5	1.7
TAM5	317.1	300.9	307.8	269.6
TAM8	93.3	81.8	83.8	77.4
TAM12	660.9	660.6	624.1	480.9
PHOENIX10	68.6	43.7	30.4	10.7
PHOENIX30	*	*	*	495.1
AMMO 4H	NA	† 1,241.3	253.9	288.4

• NA indicates problem instance not run.

† This problem was run on an IBM 3081K under the MVS operating system using VS FORTRAN 1.4.1 in a 32 megabyte virtual machine using an advanced starting solution. The solution time shown is adjusted to account for the approximate difference in computing speed between the IBM 3081K and the IBM 3033/AP.

TABLE 9.4: Number of Elements in Explicit Transformation Kernel Representation of Optimality or at Failure

	X	GUB	PNET	GNET
GTE	11,571	221	214	196
INVEST	8,380	682	654	521
TANKER	2,256	415	312	81
GAS PN A	*132,995	*130,883	35,395	65,558
GAS PN C	21,912	14,765	260	364
GAS PN E	*174,482	111,821	4,808	4,312
GAS GN A	*132,995	*130,883	*127,780	65,081
GAS GN C	17,966	8,017	1,358	608
GK 2	8,280	2,719	221	184
GK 3	13,991	7,771	388	310
GK 4	*110,334	79,646	6,532	9,155
ODSM1	1,322	861	259	421
ODSM2	418	3	3	3
TAM1	1,713	1,231	841	1,165
TAM2	3,042	2,078	1,444	1,610
TAM3	10,430	8,326	5,305	6,525
TAM4	1,489	1,302	867	1,332
TAM5	29,661	23,790	15,689	18,737
TAM8	19,545	14,886	8,351	11,182
TAM12	37,716	28,935	16,097	20,486
PHOENIX10	73,737	38,705	21,879	1,458
PHOENIX30	*153,781	*151,746	*148,348	11,523
AMMO 4H	NA	122,085	23	23

TABLE 9.5: Summary of the Number of Binding Explicit Constraints at Optimality

	Total Binding	Percent	GUB	PNET	GNET
GTE	552	57.5	19	18	18
INVEST	758	56.7	199	194	162
TANKER	50	60.2	31	30	9
GAS PN A	3,051	44.6	*	292	319
GAS PN C	2,337	61.6	849	66	91
GAS PN E	897	75.8	572	88	81
GAS GN A	3,045	44.5	*	*	352
GAS GN C	2,331	58.7	863	402	88
GK 2	1,950	51.1	1,011	93	90
GK 3	2,912	50.8	1,360	140	141
GK 4	4,119	53.9	2,477	363	356
ODSM1	297	9.8	53	49	47
ODSM2	448	75.4	219	215	0
TAM1	46	50.5	44	32	35
TAM2	87	48.3	77	51	61
TAM3	148	55.0	131	97	107
TAM4	121	57.3	110	59	81
TAM5	252	57.5	228	170	186
TAM8	276	65.7	238	140	174
TAM12	413	65.7	355	208	252
PHOENIX10	1,085	67.1	1,083	1,082	77
PHOENIX30	3,477	80.8	*	*	109
AMMO 4H	2,889	20.7	2,882	7	7

TABLE 9.6: $f(s, t)$ in Megabyte · seconds

	X	GUB	PNET	GNET
GTE	84	47	54	64
INVEST	25	16	19	20
TANKER	112	15	13	42
GAS PN A	*	*	926	1,771
GAS PN C	73	58	27	31
GAS PN E	*	509	9	14
GAS GN A	*	*	*	1,774
GAS GN C	62	45	30	34
GK 2	27	19	18	21
GK 3	90	74	60	66
GK 4	*	1,368	309	387
ODSM1	4	6	5	7
ODSM2	49	47	48	53
TAM1	0.1	0.1	0.1	0.1
TAM2	1	1	1	1
TAM3	14	10	9	12
TAM4	0.1	0.1	0.1	0.1
TAM5	750	662	614	622
TAM8	133	106	95	110
TAM12	2,376	2,213	1,882	1,632
PHOENIX10	169	65	32	7
PHOENIX30	*	*	*	693
AMMO 4H	NA	9,093	933	1194

X. CONCLUSIONS

We have presented three dynamic row factorization algorithms for solving large-scale linear programs. Although each may be used to solve any LP instance, each is designed to exploit a particular model row structure: generalized upper bounds (GUB), pure network rows (PNET) or generalized network rows (GNET).

Previous research by others generally suggests that specialized algorithms such as those presented here are useful only when the factored structure completely dominates the structure of the model instance. There are reports of algorithms for solving problems having a single unfactored (explicit) constraint (Hultz and Klingman [1978], Klingman and Russell [1978]). When implementations are reported, problem suites are limited to instances having a very small number of explicit constraints, typically in the range from one to twenty (Chen and Saigal [1977], Glover, Karney, Klingman and Russell [1978], Glover and Klingman [1981]). The consensus seems to be that such algorithms are appropriately viewed as specialized algorithms, useful only for solving very special problem instances.

Our experience strongly refutes this view. We find the performances of our implementations of the dynamic factorization algorithms are competitive with that of a commercial-quality optimization system on every model instance we have tested. This is particularly remarkable for two reasons. First, our test suite consists of models developed by skilled modelers specifically to exploit the capabilities and characteristics of the solver with which our implementations are competing. Second, we must select the row factorizations without the benefit of guidance from the

modeler, relying instead on useful but imperfect heuristics. Despite these computational handicaps, our tests show our implementations to be at least as efficient as a well-respected commercial-quality optimization system.

Our development has stressed the similarity between the algorithms and the natural extension which leads from one to the next. This is in contrast to the development that has been reported for similar, non-dynamic algorithms (e.g., Dantzig and Van Slyke [1967], Klingman and Russell [1978] and Hultz and Klingman [1978]) in which the specifics of the individual algorithm obscure the generality of the approach. The conceptual difference between our algorithms is seen to be largely isolated to the structure of a single algebraic entity, the factored kernel. By abstracting the structure of the factored kernel and concentrating on the general algorithm design, we demonstrate the versatility and flexibility of this approach.

We are gratified to find that the modularity suggested by the algorithmic development can be realized in an implementation design. We succeed in developing a software suite which displays a "single-system image". The modularity of the algorithm allows the definition of an "abstract data type" (see, e.g., Aho, Hopcroft and Ullman [1974]) which isolates the data structures and update procedures for the factored kernel from the rest of the implementation. Each factorization is seamlessly integrated within the system design, presenting a single design image.

The early 1980's produced a great deal of research in the area of automatic identification of special structure in LP models (see, e.g., Gunawardane and Schrage [1977], Glover [1980], Schrage [1981], Brown, McBride and Wood [1985] and Bixby and Fourer [1986]). We have incorporated the most useful of these ideas into our implementation, and we have what we believe to be the first implementation which supports the automatic identification of factored row sets. This capability may

be used to identify new factored structure or to validate or augment a modeler-provided recommendation. Our computational experience indicates that while this approach is not as promising as perhaps first envisioned, it is nonetheless a valuable tool. When faced with the choice of either solving an unfactored model instance or automatically identifying a factored structure and then using the corresponding solver, our results show that the latter is nearly always to be preferred. Our results seem to suggest, however, that in addition to quantity of factored rows, the issue of quality of factored rows exerts influence on the performance of the factorization algorithms. While not well understood, it is clear that the myopic approach of our heuristics is no substitute for the modeler's guidance in identifying factored structure.

Several areas suggest themselves for further research. Certainly additional factored structures can be examined. For example, one approach for treating factored column structures ("complicating columns") is to allow the partitioning of rows into the categories "factored" and "explicit" to vary as the algorithm progresses. That is, allow factored row set membership to be determined with respect to the column structure of the currently nonbasic variables rather than with respect to the column structure of all problem variables. While conceptually simple, such a generalization seems to present significant algorithmic challenges.

General algorithms are sometimes useful in specialized contexts. For example, processing networks (Koene [1982]) are network models which allow proportional flow restrictions on the arcs entering or leaving some nodes. One formulation of such a model results in a pure or generalized network structure with a set of complicating columns. Chen and Engquist [1986] propose a primal partitioning algorithm for solving processing network problems. An alternate formulation yields a pure or

generalized structure with complicating rows, and we note that this is precisely the structure we seek for our network factorizations.

The multicommodity capacitated transshipment problem (MCTP) has been the subject of much research over the years, and a number of specialized algorithms (see, e.g., Assad [1978] or Kennington [1978]) have been proposed to solve it. Adopting a general perspective, MCTP may be viewed either as a GUB model or as a pure network model with side constraints, and either view might be preferred depending upon the dimensions of the particular instance under consideration. Our computational experience indicates that the pure network factorization algorithm offers a powerful technique for solving MCTP. As an experiment, we customized our (PN) implementation to exploit the special structure of the side constraints in MCTP. It is interesting to note that in our scientific computing environment, we observed no difference in solution times between the customized version of (PN) and the original implementation.

Finally, all the approaches we have considered assume the prior existence of a specific structure in the factored rows which in turn determines the structure of the factored kernel. An extension of this general approach is to relax the requirement for strict conformance to a specific structure. Instead, we might allow the factored row structure to be "nearly" homogeneous. For example, we may allow a small number of complicating columns to disturb what is otherwise a factored pure network row structure. We then expect the structure of the factored kernel to be dominated by that induced by the predominant row structure, with only occasional complications due to the exceptional row structure. We allow for this exceptional structure in the factored kernel by identifying it "on-the-fly" as the algorithm progresses, and treating it in an appropriate manner. This approach may be thought of as a hybrid between the factored method developed here and dynamic basis triangulation

methods (see, e.g., Hellerman and Rarick [1971] and [1972], Saunders [1976] and McBride [1980]).

Dynamic extrinsic factorization is subsumed if we activate functions in the update analogous to the secondary exchanges now employed. Essentially all that has to be done is ensure that successive factored components retain their stipulated special structure. In our estimation, this will only be justified in cases where the model structure is amenable, and quite likely will require some model-specific features to perform well on difficult models. We have limited our experimentation to those static extrinsic cases which are believed to be most useful.

LIST OF REFERENCES

1. Aho, A. V., Hopcroft, J. E. and Ullman, J. D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Co., Menlo Park, California, 1974.
2. Assad, A., "Multicommodity Network Flows—A Survey," *Networks*, Vol. 8, 1978, pp. 37-91.
3. Avery, W., Brown, G. G., Rosenkranz, J. A. and Wood, R. K., "A Supply Optimization System for Natural Gas Distribution Companies," (in preparation), 1989.
4. Bartels, R. H. and Golub, G. H., "The Simplex Method of Linear Programming Using LU Decomposition," *Communications of the ACM*, Vol. 12, 1969, pp. 266-268.
5. Bradley, G. H., "Optimization of Capital Portfolios," *Proceedings of the National Communications Forum* 86, 1986, pp. 11-17.
6. Bradley, G. H., Brown, G. G. and Graves, G. W., "Design and Implementation of Large-Scale Primal Transshipment Algorithms," *Management Science*, Vol. 24-1, 1977, pp. 1-34.
7. Benders, J. F., "Partitioning Procedure for Solving Mixed-Variables Programming Problems," *Numerische Mathematik*, Vol. 4, 1962, pp. 238-252.
8. Bennett, J. M., "An Approach to Some Structured Linear Programming Problems," *Operations Research*, Vol. 14-4, 1966, pp. 636-645.
9. Bixby, R. E. and Fourer, R., *Finding embedded network rows in linear programs I: Extraction heuristics*, Bonn University, Oekonometrie und Operations Research, Report No. 86437-OR, July, 1986.
10. Brown, G. G. and Graves, G. W., "XS Mathematical Programming System" perpetual working paper, 1975.
11. Brown, G. G., Graves, G. W. and Ronen, D., "Scheduling Ocean Transportation of Crude Oil," *Management Science*, Vol. 33-3, 1987, pp. 335-346.
12. Brown, G. G. and McBride, R. D., "Solving Generalized Networks," *Management Science*, Vol. 30-12, 1984, pp. 1497-1523.
13. Brown, G. G., McBride, R. D. and Wood, R. K., "Extracting Embedded Generalized Networks for linear programming problems," *Mathematical Programming Study*, 32, 1985, pp. 11-31.
14. Brown, G. G. and Thomen, D., "Automatic identification of generalized upper bounds in large-scale optimization models," *Management Science*, No. 26-11, 1980, pp. 1166-1184.

15. Brown, G. G. and Wright, W., "Automatic Factorization of Embedded Structure in Large-Scale Optimization Models," *Mathematical Programming*, 24, 1984, pp. 41-46.
16. Charnes, A. and Lemke, C. E., Computational Theory of Linear Programming, I: The Bounded Variables Problem, *ONR Research Memorandum 10*, Graduate School of Industrial Administration, Carnegie Institute of Technology, Pittsburgh, Pennsylvania. 1952.
17. Chen, C. and Engquist, M., "A Primal Simplex Approach to Pure Processing Networks," *Management Science*, Vol. 32-12, 1986, pp. 1582-1598.
18. Chen, S. and Saigal, R., "A Primal Algorithm for Solving a Capacitated Network Flow Problem with Additional Linear Constraints," *Networks*, Vol. 7, 1977, pp. 59-79.
19. Clemence, R. D. Jr., Teufert, W. R., Brown, G. G. and Wood, R. K., "Phoenix: Developing and Evaluating Army Aviation Modernization Policies Using Mixed Integer Linear Programming," *27th U. S. Army Operations Research Symposium*, Fort Lee, Virginia, October 12-13, 1988.
20. Dantzig, G. B., Notes on Linear Programming: Parts VIII, IX. X-Upper Bounds, Secondary Constraints, and Block Triangularity in Linear Programming, *Research Memorandum RM-1367*, The Rand Corporation, Santa Monica, California, October, 1954.
21. Dantzig, G. B., *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey, 1963.
22. Dantzig, G. B. and Van Slyke, R. M., "Generalized Upper Bounding Techniques," *Journal of Computer and System Sciences*, Vol. 1, 1967, pp. 213-226.
23. Dantzig, G. B. and Wolfe, P., "Decomposition principal for linear programming," *Operations Research*, Vol. 8-1, 1960, pp. 101-111.
24. Elam, J. F., Glover, F. and Klingman, D., "A Strongly Convergent Primal-Simplex Algorithm for Generalized Networks," *Mathematics of Operations Research*, 4-1, 1979, pp. 39-59.
25. Forrest, J. J. H. and Tomlin, J. A., "Updating Triangular Factors of the Basis to Maintain Sparsity in the Product Form Simplex Method," *Mathematical Programming*, Vol. 2, 1972, pp. 262-278.
26. Garey, M. R. and Johnson, D. S., *Computers and Intractability*, W. H. Freeman and Co., San Francisco, California, 1979.
27. Geoffrion, A. M. and Graves, G. W., "Multicommodity Distribution System Design by Benders Decomposition," *Management Science*, Vol. 29-5, January, 1974, pp. 822-844.
28. Glover, F., "Transformations enlarging the network portion of a class of LP/embedded generalized networks," *MSRS 80-1*. University of Colorado, Boulder, Colorado, April, 1980.
29. Glover, F., Hultz, J., Klingman, D. and Stutz, J., "A New Computer-Based Planning Tool," *Research Report CCS 289*, Center for Cybernetic Studies, University of Texas at Austin, Austin, Texas, 1977.

30. Glover, F., Hultz, J., Klingman, D. and Stutz, J., "Generalized Networks: A Fundamental Computer-Based Planning Tool," *Management Science*, Vol. 24-12, 1978, pp. 1209-1220.
31. Glover, F., Karney, D. and Klingman, D., "The Augmented Predecessor Index Method for Locating Stepping Stone Paths and Assigning Dual Prices in Distribution Problems," *Transportation Science*, Vol. 6-2, 1972, pp. 171-179.
32. Glover, F., Karney, D., Klingman, D. and Napier, A., "A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems," *Management Science*, Vol. 20-5, 1974, pp. 793-813.
33. Glover, R., Karney, D., Klingman, D. and Russell, R., "Solving Singly Constrained Transshipment Problems," *Transportation Science*, Vol. 12-4, 1978, pp. 277-297.
34. Glover, F. and Klingman, D., "A Note on Computational Simplifications in Solving Generalized Transportation Problems," *Transportation Science*, Vol. 7-4, 1973, pp. 351-361.
35. Glover, F. and Klingman, D., "The Simplex SON Algorithm for LP/Embedded Network Problems," *Mathematical Programming Study*, 15, 1981, pp. 148-176.
36. Glover, F., Klingman, D. and Stutz, J., "Extensions of the Augmented Predecessor Index Method to Generalized Network Problems," *Transportation Science*, Vol. 7-4, 1973, pp. 377-384.
37. Glover, F., Klingman, D. and Stutz, J., "Augmented Threaded Index Method for Network Optimization," *INFOR*, Vol. 12-3, 1974, pp. 293-298.
38. Golub, G. H. and Van Loan, C. F., *Matrix Computations*, The Johns Hopkins University Press, Baltimore, Maryland, 1983.
39. Graves, G. W., "A Complete Constructive Algorithm for the General Mixed Linear Programming Problem," *Naval Research Logistics Quarterly*, 12-1, 1965, pp. 1-14.
40. Graves, G. W. and McBride, R. D., "The Factorization Approach to Large-Scale Linear Programming," *Mathematical Programming*, Vol. 10, 1976, pp. 91-110.
41. Graves, G. W., "Mathematical Programming", in preparation, 1989.
42. Greenberg, H. J. and Rarick, D. C., "Determining GUB sets via a invert agenda algorithm," *Mathematical Programming*, Vol. 7, 1974, pp. 240-244.
43. Gunawardane, G. and Schrage, L., "Identification of special structure constraints in linear programs," University of Chicago, Chicago, Illinois, 1977.
44. Harrison, T. P., Bradley, G. H. and Brown, G. G., "Capital allocation and project selection via decomposition," presented at CORS/TIMS/ORSA meeting, Vancouver, British Columbia, Canada, May, 1989.

45. Hartman, J. K. and Lasdon, L. S., "A generalized upper bounding method for doubly coupled linear programs," *Technical Memorandum No. 140*, June, 1970.
46. Hartman, J. K. and Lasdon, L. S., "A Generalized Upper Bounding Algorithm for Multicommodity Network Flow Problems," *Networks*, 1, 1972, pp. 333-354.
47. Helgason, R. V. and Kennington, J. L., "A Product Form Representation of the Inverse of a Multicommodity Cycle Matrix," *Networks*, Vol. 7, 1977, pp. 297-322.
48. Hellerman, E. and Rarick, D., "Reinversion and the preassigned pivot procedure," *Mathematical Programming*, Vol. 1, 1971, pp. 195-216.
49. Hellerman, E. and Rarick, D., "The partitioned preassigned pivot procedure (p^4)," in: Rose, D. J. and Willoughby, P. A., eds., *Sparse Matrices and their Applications*, Plenum Press, New York, New York, 1972, pp. 67-76.
50. Hultz, J. and Klingman, D., "Solving Constrained Generalized Network Problems," *Research Report CCS 257*, Center for Cybernetic Studies, University of Texas at Austin, Austin, Texas, 1976.
51. Hultz, J. and Klingman, D., "Solving Singularly Constrained Generalized Network Problems," *Applied Mathematics and Optimization*, Vol. 4, 1978, pp. 103-119.
52. Jackson, J. A., *A Taxonomy of Advanced Linear Programming Techniques and the Theater Attack Model*, Master's Thesis, Air Force Institute of Technology, Air University, Wright-Patterson Air Force Base, Ohio, 1989.
53. Johnson, E. L., "Networks and Basic Solutions," *Operations Research*, Vol. 14-4, 1966, pp. 619-623.
54. Kaul, R. N., "An Extension of Generalized Upper Bounding Techniques for Linear Programming," *ORC Report No. 65-27*, Department of Operations Research, University of California at Berkeley, Berkeley, California, 1965.
55. Kennington, J. L., "Solving Multicommodity Transportation Problems Using a Primal Partitioning Simplex Technique," *Naval Research Logistics Quarterly*, Vol. 24-2, 1977, pp. 309-325.
56. Kennington, J. L., "A Survey of Linear Cost Multicommodity Network Flows," *Operations Research*, Vol. 26, 1978, pp. 209-236.
57. Klee, V. and Minty, G. J., "How good is the simplex algorithm?," in: Shisha, O. (ed.), *Inequalities-III*, Academic Press, New York, New York, 1972, pp. 159-172.
58. Klingman, D. and Russell, R., "On Solving Constrained Transportation Problems," *Operations Research*, Vol. 23-1, 1975, pp. 91-107.
59. Klingman, D. and Russell, R., "A Streamlined Simplex Approach to the Singly Constrained Transportation Problem," *Naval Research Logistics Quarterly*, 25-4, 1978, pp. 581-696.

60. Koene, J., *Minimal Cost Flow in Processing Networks, a Primal Approach*, Ph.D. Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 1982.
61. Magnanti, T. L., "Optimization for Sparse Systems," in: Bunch, J. R. and Rose, D. J., eds., *Sparse Matrix Computations*, Academic Press, New York, New York, 1976, pp. 147-176.
62. McBride, R. D., "Factorization in Large-Scale Linear Programming," *Working Paper No. 22*, University of California, Los Angeles, California, 1972.
63. McBride, R. D., "A Bump Triangular Dynamic Factorization Algorithm for the simplex method," *Mathematical Programming*, Vol. 18, 1980, pp. 49-61.
64. McBride, R. D., "Solving Embedded Generalized Network Problems," *European Journal of Operational Research*, 21, 1985, pp. 82-92.
65. McBride, R. D., Private Communication, 1989.
66. Might, R. J., "Decision Support for Aircraft and Munitions Procurement," *Interfaces*, 17-5, September-October, 1987, pp. 55-63.
67. Murtagh, B. A. and Saunders, M. A., "MINOS User's Guide," *Technical Report SOL 77-9*, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California, 1977.
68. Murtagh, B. A., *Advanced Linear Programming: Computation and Practice*, McGraw-Hill International Book Company, New York, New York, 1981.
69. Reid, J. K., "A Sparsity-Exploiting Variant of the Bartels-Golub Decomposition for Linear Programming," *Report CSS 20*, Computer Science and Systems Division, A.E.R.E., Harwell, England, 1975.
70. Rosen, J. B., "The gradient projection method for nonlinear programming-Part I. linear constraints," *Journal of the Society of Applied Mathematics*, Vol. 8-1, 1960, pp. 181-217.
71. Rosen, J. B., "Primal Partition Programming for Block Diagonal Matrices," *Numerical Mathematics*, Vol. 6, 1964, pp. 250-260.
72. Saunders, M. A., "A Fast, Stable Implementation of the Simplex Method Using Bartels-Golub Updating," in: Bunch, J. R. and Rose, D. J., eds., *Sparse Matrix Computations*, Academic Press, New York, New York, 1976, pp. 213-226.
73. Schrage, L., "Implicit representation of variable upper bounds in linear programming," *Mathematical Programming*, Vol. 4, 1975, pp. 118-132.
74. Schrage, L., "Implicit representation of generalized variable upper bounds in linear programming," *Mathematical Programming*, Vol. 14, 1978, pp. 11-20.
75. Schrage, L., "Some comments on hidden structure in linear programs," in: Greenberg, H. J. and Maybee, I., eds., *Computer-assisted Analysis and Model Simplification*, Academic Press, New York, New York, 1981, pp. 389-395.
76. Srinivasan, V. and Thompson, J. L., "Benefit-cost analysis of coding techniques for the primal transportation algorithm," *Journal of the Association for Computing Machinery*, Vol. 20-1, 1973, pp. 194-213.

77. Staniec, C. J., *Design and Solution of an Ammunition Distribution Model by a Resource-Directive Multicommodity Network Flow Algorithm*, Master's Thesis, Naval Postgraduate School, Monterey, California, 1984.
78. Todd, M. J., "Large-scale linear programming: Geometry, working bases and factorization," *Mathematical Programming*, Vol. 26-1, 1983, pp. 1-20.
79. Tomlin, J. A., "Survey of computational methods for solving large scale systems," *Technical Report 72-25*, Stanford University, Stanford, California, 1972.
80. Wood, R. K., Private Communication, 1989.

INITIAL DISTRIBUTION LIST

		No. of Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Professor Gerald G. Brown, Code 55Bw Department of Operations Research Naval Postgraduate School Monterey, California 93943-5000	20
4.	Commander Michael P. Olson, Code 55 Department of Operations Research Naval Postgraduate School Monterey, California 93943-5000	1
5.	Professor Richard D. McBride Decision Systems Department School of Business University of Southern California Los Angeles, California 90089-1421	1
6.	Chief of Naval Operations (OP-81) Department of the Navy Washington, D. C. 20350	1
7.	Office of Naval Research Mathematics (Code 411) 800 N. Quincy Street Arlington, Virginia 22217	1